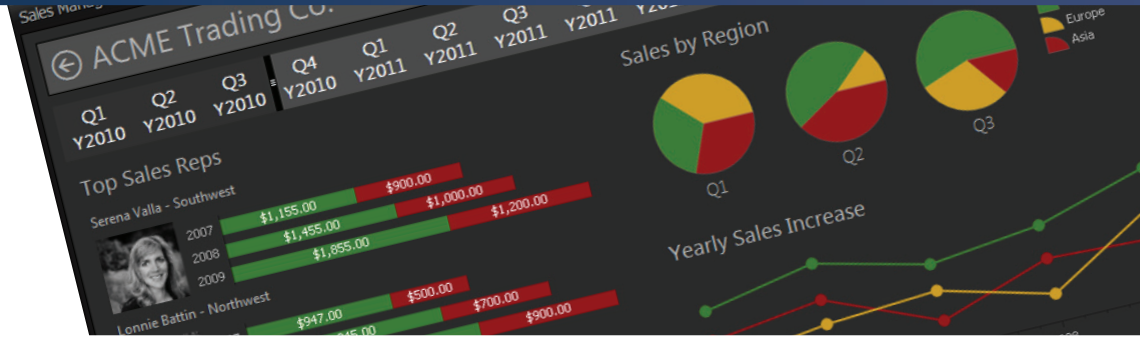
Microsoft
GOLD CERTIFIED
Partner



DXv2 is here

Simply Stunning.

Today's users expect beautiful apps in every part of their lives, from work to home. Now, with DevExpress tools, you can apply sophisticated themes and incorporate elegant Office-inspired controls into your designs. DXv2 delivers the tools you need to inspire and be inspired.

Download the 30-day free trial to experience the next generation of developer productivity tools at **www.DevExpress.com**

DevExpress

THE MICROSOFT JOURNAL FOR DEVELOPERS

JANUARY 2012 VOL 27 NO 1

msdn magazine

WINDOWS PHONE

PLUS:

COLUMNS

Microsoft

Write Once, Experience Many

check out infragistics.com/jquery

NetAdvantage® for jQuery

TREE

Simplify the look of hierarchical data, while offering the experience, design and functionality your users will love!

BUSINESS CHARTING

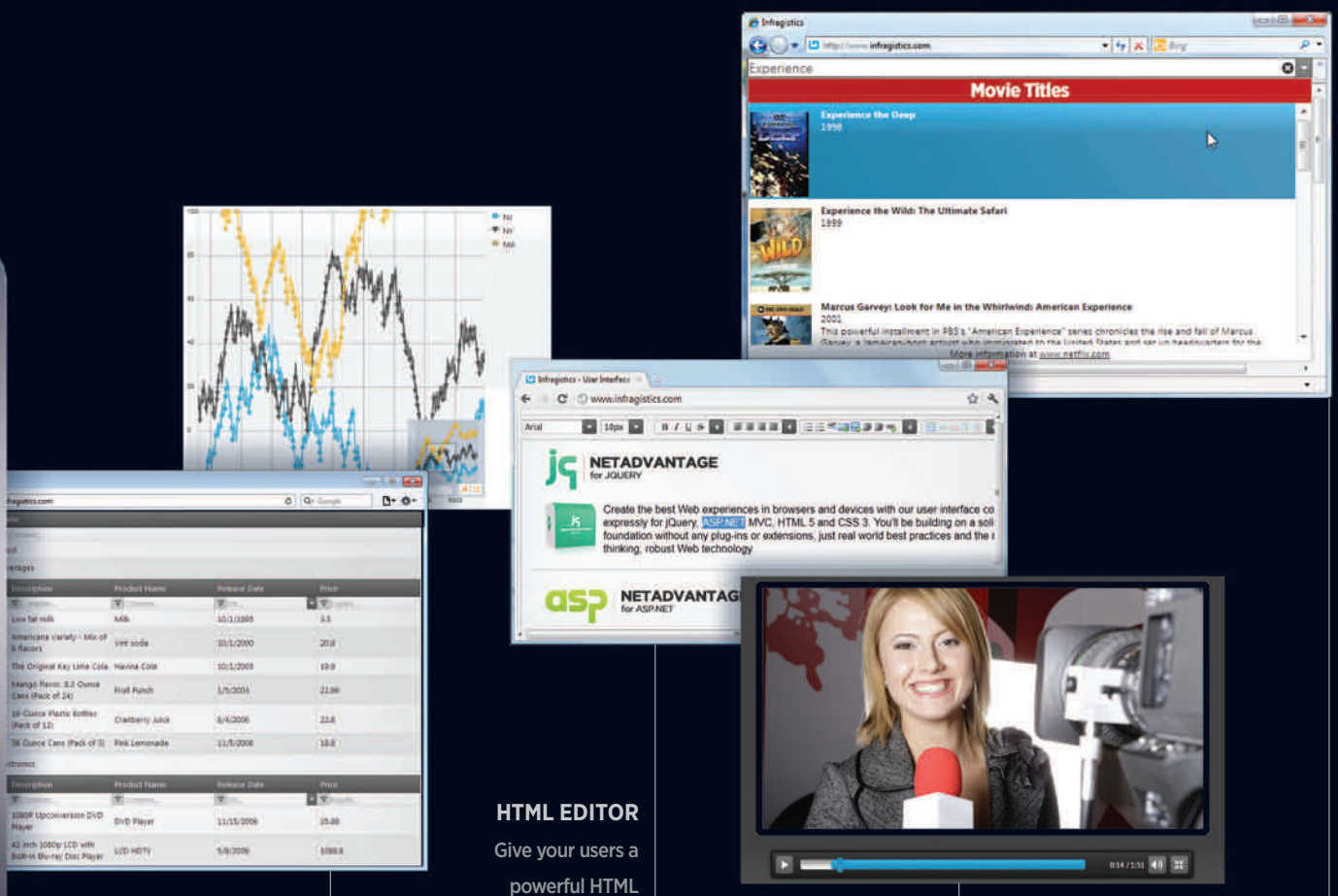
Combine interactive Outlook style grids with rich business charting to deliver a complete portable solution.

INFRAGISTICS

DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales 800 231 8588 • Infragistics Europe Sales +44 (0) 800 298 9055 • Infragistics India +91 80 4151 8042 • @infragistics

Copyright 1996-2011 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc.



HIERARCHICAL GRID

An expandable data grid that presents multiple parent-child relationships is the backbone of your data application.

HTML EDITOR

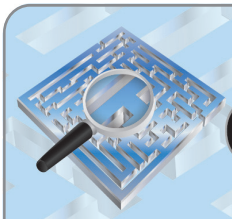
Give your users a powerful HTML editing experience by incorporating the jQuery WYSIWYG editing tool.

VIDEO PLAYER

When a user finds what they want to watch, our HTML5 video player adds streaming video right into your own apps.

COMBO

The fully featured combo box control offers intuitive auto-suggest, auto-complete and auto-filtering built in.



dtSearch®

Instantly Search Terabytes of Text

"Bottom line: dtSearch manages a terabyte of text in a single index and returns results in less than a second" *InfoWorld*

"Covers all data sources ... powerful Web-based engines" *eWEEK*

"Lightning fast ... performance was unmatched by any other product" *Redmond Magazine*

For hundreds more reviews and developer case studies, see www.dtSearch.com

Highlights hits in a wide range of data, using dtSearch's *own* file parsers and converters

- Supports MS Office through 2010 (Word, Excel, PowerPoint, Access), OpenOffice, ZIP, HTML, XML/XSL, PDF and more
- Supports Exchange, Outlook, Thunderbird and other popular email types, including nested and ZIP attachments
- Spider supports static and dynamic web data like ASP.NET, MS SharePoint, CMS, PHP, etc.
- API for SQL-type data, including BLOB data

25+ full-text & fielded data search options

- Federated searching
- Special forensics search options
- Advanced data classification objects

APIs for C++, Java and .NET through 4.x

- Native 64-bit and 32-bit Win / Linux APIs; .NET Spider API
- Content extraction only licenses available

Desktop with Spider | **Web with Spider**

Network with Spider | **Engine for Win & .NET**

Publish (portable media) | **Engine for Linux**

Ask about fully-functional evaluations!

The Smart Choice for Text Retrieval® since 1991

www.dtSearch.com • 1-800-IT-FINDS

msdn® magazine

JANUARY 2012 VOLUME 27 NUMBER 1

LUCINDA ROWLEY Director
KIT GEORGE Editorial Director/mmeditor@microsoft.com
PATRICK O'NEILL Site Manager

MICHAEL DESMOND Editor in Chief/mmeditor@microsoft.com
DAVID RAMEL Technical Editor
SHARON TERDEMAN Features Editor
WENDY GONCHAR Managing Editor
KATRINA CARRASCO Associate Managing Editor

SCOTT SHULTZ Creative Director
JOSHUA GOULD Art Director

CONTRIBUTING EDITORS Dino Esposito, Joseph Fultz, Kenny Kerr, Julie Lerman, Dr. James McCaffrey, Ted Neward, Charles Petzold, David S. Platt

Redmond Media Group

Henry Allain President, Redmond Media Group
Matt Morollo Vice President, Publishing
Doug Barney Vice President, Editorial Director
Michele Imgrund Director, Marketing
Tracy Cook Online Marketing Director

ADVERTISING SALES: 508-532-1418/mmorollo@1105media.com

Matt Morollo VP, Publishing
Chris Kourtoglou Regional Sales Manager
William Smith National Accounts Director
Danna Vedder Microsoft Account Manager
Jenny Hernandez-Asandas Director Print Production
Serena Barnes Production Coordinator/msdnadproduction@1105media.com

1105 MEDIA, INC.

Neal Vitale President & Chief Executive Officer
Richard Vitale Senior Vice President & Chief Financial Officer
Michael J. Valenti Executive Vice President

Christopher M. Coates Vice President, Finance & Administration
Erik A. Lindgren Vice President, Information Technology & Application Development
David F. Myers Vice President, Event Operations

Jeffrey S. Klein Chairman of the Board

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: *MSDN Magazine*, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o *MSDN Magazine*, 4 Venture, Suite 150, Irvine, CA 92618.

Legal Disclaimer: The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

Corporate Address: 1105 Media, Inc., 9201 Oakdale Ave., Ste 101, Chatsworth, CA 91311, www.1105media.com

Media Kits: Direct your Media Kit requests to Matt Morollo, VP Publishing, 508-532-1418 (phone), 508-875-6622 (fax), mmorollo@1105media.com

Reprints: For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International, Phone: 212-221-9595, E-mail: 1105reprints@parsintl.com, www.magreprints.com/QuickQuote.asp

List Rental: This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Merit Direct. Phone: 914-368-1000; E-mail: 1105media@meritdirect.com; Web: www.meritdirect.com/1105

All customer service inquiries should be sent to MSDNmag@1105service.com or call 847-763-9560.

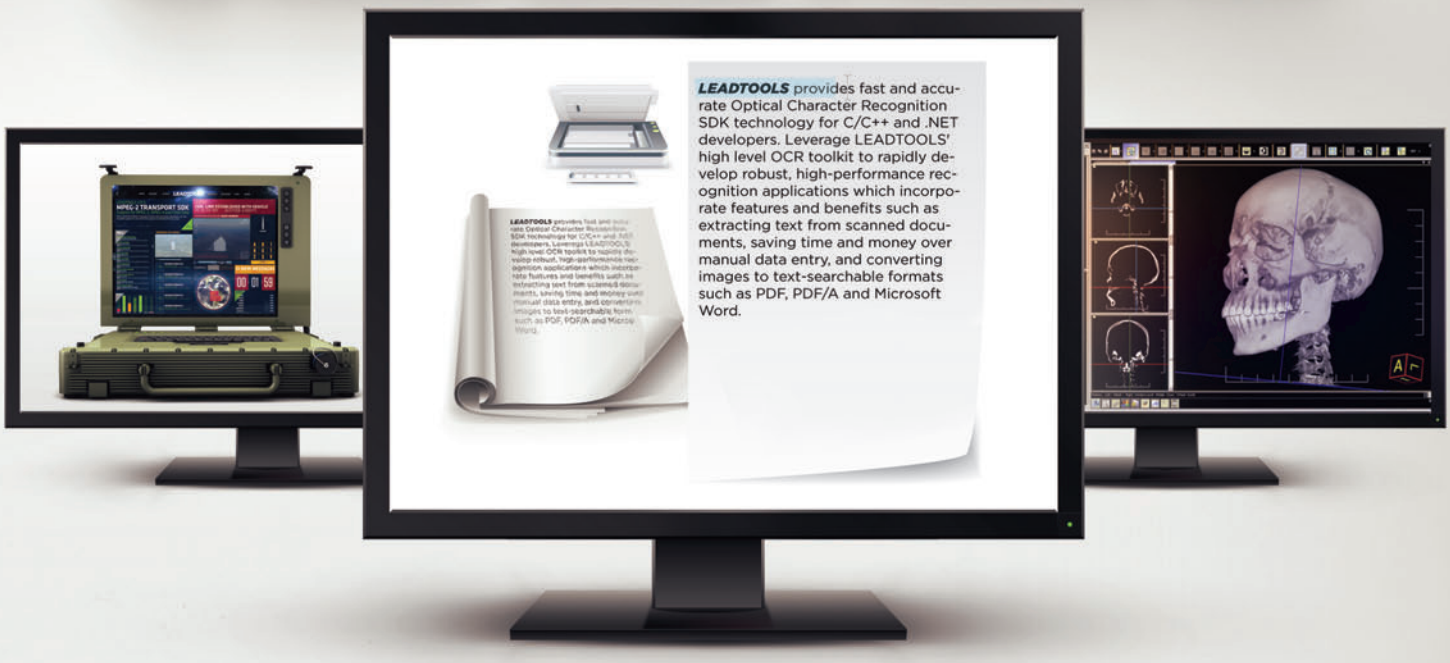
Microsoft

BPA WORLDWIDE

Printed in the USA

LEADTOOLS

THE WORLD LEADER IN IMAGING SDKs



LEADTOOLS PROVIDES DEVELOPERS EASY ACCESS TO DECADES OF EXPERTISE IN DEVELOPING COLOR, GRAYSCALE, DOCUMENT, MEDICAL, VECTOR AND MULTIMEDIA IMAGING TECHNOLOGY. DEVELOP WITH LEADTOOLS TO ELIMINATE MONTHS OF RESEARCH AND DEVELOPMENT AND ADD ESSENTIAL FUNCTIONALITY TO YOUR APPLICATION WITHOUT SACRIFICING QUALITY OR PERFORMANCE.

OCR/ICR/OMR	BARCODE	PDF & PDF/A	FORMS RECOGNITION & PROCESSING
DOCUMENT CLEANUP & PREPROCESSING	ANNOTATIONS	150 + FILE FORMATS	VIEWER CONTROLS
DICOM	PACS	IMAGE PROCESSING	SCANNING
MEDICAL 3D	MEDICAL WORKSTATION	MEDICAL WEB VIEWER	VIRTUAL PRINTER
MPEG-2 TRANSPORT STREAM	MULTIMEDIA PLAYBACK & CAPTURE	DVD & DVR	DIRECTSHOW CODECS

C++, .NET, SILVERLIGHT, WINDOWS PHONE,

C DLL, WCF SERVICES, WF ACTIVITIES, WPF, ASP.NET, COM & CLOUD

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

EDITOR'S NOTE

MICHAEL DESMOND

A Quarter Century and Counting

Twenty-five years is a long time, especially in the software development business. So it's hard to believe that Charles Petzold, author of the UI Frontiers column, has been contributing to *MSDN Magazine* and its predecessor *Microsoft Systems Journal* (*MSJ*) since this publication launched in 1986.

Petzold's contributions to the magazine stretch back to the inaugural issue of *MSJ* in October 1986, and in December of that year he wrote the first-ever magazine article on Windows programming, "A Step-by-Step Guide to Building Your First Windows Application." He then wrote one of the first books on Windows development—"Programming Windows" (Microsoft Press, 1988)—which became a definitive resource for Windows programmers. Yeah, you could say Petzold got in on the ground floor of that whole Windows thing.

What's truly remarkable is that, after all these years, Petzold is still here. From Win16 to Win32, through four-plus iterations of the Microsoft .NET Framework, and most recently the emergence of the Windows Runtime, the two constants at *MSDN Magazine* have been: change and Charles Petzold.

The success of *MSDN Magazine* might seem obvious today, but at the time it was hardly a sure thing. No one had ever published a magazine quite like this before. The whole thing got rolling when Jon Lazarus, a former executive at Ziff-Davis, left the company to publish *MSJ* under contract. As Petzold tells it, Lazarus knew him as a writer at *PC Magazine* who was "doing silly stuff with Windows." And despite a late change of strategy, that was exactly what the fledgling publication needed.

"Originally the magazine was supposed to be exclusively about Windows programming, but they chickened out because there was no indication that Windows would be successful," Petzold recalls. "They took a safer route that it would be about programming for all Microsoft operating systems. And because Microsoft was always rather enamored of IBM, and IBM published *IBM Systems Journal*, they called it *Microsoft Systems Journal*."

The first issues were produced in the Manhattan office that Lazarus shared with a TV talent agent. The space, Petzold says, was "filled with stacks of videotapes," and was located not too far from the offices of Ziff-Davis' *PC Magazine*. That proximity enabled a robust back-and-forth between *PC Magazine* and *MSJ* that helped keep the publication vital.

"That social connection continued for years: *PC Magazine* people and *MSJ* people would frequently hang out together at industry events such as Comdex, and get together for parties and dinners in New York City," Petzold says, adding, "And sometimes editors would hop from one of the magazines to the other. Tony Rizzo went from *MSJ* to *PC Magazine*, and Sharon Terdeman, who works for *MSDN Magazine* now, I originally knew when she was at *PC Magazine*."

By the mid-1990s, Petzold says, that social interaction had "pretty much disintegrated. Or maybe these dinners are still happening and they just stopped inviting me!"

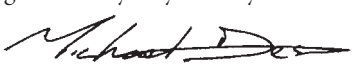
A lot more than dinner has changed since Petzold came on board. "Gosh, in 1986 there were still people arguing that the personal computer didn't need graphics," Petzold says. "Twenty-five rows of 80 characters of text were just fine for those folks."

How times change. Today, Petzold describes the emergence of hand-held touch devices as "a third revolution" of personal computing (after the GUI and the Internet). It's an area Petzold has dedicated himself to in his UI Frontiers column, which has focused largely on Windows Phone development since the mobile platform debuted. In fact, this month his column gets a new name—Touch and Go—reflecting the unique challenges and opportunities of these emergent devices.

But even as he heralds a new revolution, Petzold worries that developers, increasingly, have less and less in common with one another.

"Everybody seems to be working on something different, and it's impossible for any one person to be familiar with all these different technologies," Petzold says. "We've all become specialists. There's no longer an industry event like Comdex that virtually everybody attends, no longer books that everybody reads, no longer languages that everyone speaks.

"It's a problem, and it doesn't seem to be getting any better," Petzold continues. "But the extreme biodiversity that exists now is perhaps an indication that the art and engineering of computer programming is still in its infancy. And that suggests we need to keep our minds open—to evaluate new frameworks and programming languages, with the thought that they may actually be better than what we're using now."



Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

© 2012 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in MSDN Magazine are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, MSDN, and Microsoft logos are used by 1105 Media, Inc. under license from owner.

axosoft
800.653.0024

Everything You Need to Ship Software OnTime

Powerful bug tracking.

Manage bugs, defects, & issues with automatic notifications & alerts. Customize the tracker to match your development workflow. Track features and requirements, too—with drag and drop subitems support.

The **best** Scrum tool.

Utilize Scrum development or another agile process? OnTime is easy to adapt, and includes intelligent burndown & burnup charts, sprint & release backlogs, and the best interactive planning board out there.

A **beautiful & fast** UI.

You shouldn't spend all of your time in a software dev tool when you could be actually coding. OnTime's UI is fast and simple, and you can get what you need done right away. We think you'll really like it.

Plus: OnTime Mobile • GitHub Integration • Customer Portal & Help Desk • Built-in Scratchpad
Remote Server • Windows, Visual Studio & Eclipse Clients • Premium Hosting • SDK • and more!

Visit axosoft.com to learn more.

Free 2-user hosted license. **Forever.**

The best way to find out how OnTime can help your team is to sign up for a free 2-user license. Hosted on our reliable, secure OnTime Now! service, you can get started right away. Plus, for 30 days you can add up to 10 total users—but you'll still have 2 users forever.

Did you **know?**

OnTime is built on the Microsoft .NET framework, and it has a Microsoft SQL Server back-end. You can use OnTime as a hosted service or install it in your own environment. With OnTime's APIs your dev team can extend the functionality of OnTime inside your apps.

Find us on:

@axosoft Axosoft OnTime /axosoft

© Copyright 2002 - 2011 Axosoft LLC | All Rights Reserved
All trademarks and registered trademarks are the property of their respective owners.

axosoft OnTime11

Team up. Collaborate. Build **great software.**

DINO ESPOSITO

Enhancing the Context-Sensitive ASP.NET MVC Progress Bar

In the Web world, the term "progress bar" means too many different things to different people. Sometimes it refers to static text that simply shows that some operation is taking place somewhere. The text provides basic feedback to the user and essentially invites her to just relax and wait. Sometimes, the progress bar displays a simple animation while sending the same message to the user—please wait—which at least focuses the user's attention because users tend to follow the moving elements. A typical animation shows a graphic element that moves around a circular or linear path indefinitely. When the moving element reaches the end of the path, the animation starts over and keeps running until the underlying operation terminates and the animation is stopped programmatically. This is a fairly common pattern on, for example, most airline Web sites.

Last month, I introduced a basic version of an ASP.NET MVC framework—the SimpleProgress Framework—that allows you to quickly and effectively set up a truly context-sensitive progress bar (msdn.microsoft.com/magazine/hh580729). Context-sensitive is probably just what a progress bar should be. That is, it should be an element of the UI that progressively describes the status of an ongoing operation. The progressive display can be as simple as a sequence of messages or it can be more compelling—for example, a gauge.

In this article, I'll show how to enhance the progress bar by adding cancel capabilities. In other words, if the user interacts with the interface and cancels the operation, the framework will direct the manager of the ongoing operation to interrupt any work being done.

Canceling Ongoing Tasks

Canceling an ongoing server-side task from within a client browser is not a trivial operation. Don't be fooled by the very basic examples you'll find that just abort the client request and pretend that everything is also cleared on the server.

When you trigger a server operation via AJAX, a socket is opened that connects your browser to a remote endpoint. This connection remains open, waiting for the request to complete and return a response. The trick I discussed last month for setting up a context-sensitive progress bar used a parallel flow of requests to check a second controller method responsible for returning the status of the operation—sort of a mailbox that client and server can use to communicate.

Now suppose there's an Abort button that lets the user cancel the current server operation. What kind of code is that going to require? At the very least, you want to abort the AJAX request. If the server operation was started using the jQuery AJAX API, you can do the following:

```
xhr.abort();
```

Code download available at code.msdn.microsoft.com/mag201201CuttingEdge.

Figure 1 **Adding Abort Functionality to the Progress Framework**

```
var ProgressBar = function () {
  var that = {};

  // Store the XHR object being used.
that._xhr = null;

// Get the user-defined callback that runs after
// aborting the call.
that._taskAbortedCallback = null;

...

// Set progress callbacks.
that.callback = function (userCallback, completedCallback,
                          abortedCallback) {
  that._userDefinedProgressCallback = userCallback;
  that._taskCompletedCallback = completedCallback;
  that._taskAbortedCallback = abortedCallback;
  return this;
};

// Abort function.
that.abort = function () {
  if (_xhr !== null)
       xhr.abort();

};

...

// Invoke the URL and monitor its progress.
that.start = function (url, progressUrl) {
  that._taskId = that.createTaskId();
  that._progressUrl = progressUrl;

  // Place the AJAX call.
    xhr = $.ajax({
      url: url,
      cache: false,
      headers: { 'X-ProgressBar-TaskId': that._taskId },
      complete: function () {
        if (_xhr.status != 0) return;
        if (that._taskAbortedCallback != null)
            that._taskAbortedCallback();
        that.end();
      },
      success: function (data) {
        if (that._taskCompletedCallback != null)
            that._taskCompletedCallback(data);
        that.end();
            }
  });

  // Start the progress callback (if any set).
  if (that._userDefinedProgressCallback == null)
      return this;
  that._timerId = window.setTimeout(
    that._internalProgressCallback,
    that._interval);
};


  return that;
}
```

ComponentOne Ultimate™ delivers the tools and resources to build everything …everywhere. Whether you're a Windows, Web, or XAML developer, this ultimate dev tool collection delivers. Inside you'll find: 100s of .NET controls, OLAP data analysis controls, SharePoint Web Parts, documentation tools, LightSwitch extensions, and tools for ADO.NET Entity Framework and RIA Services. No job is too big. Bring speed, style, and functionality to your all your applications ...it is your destiny.

BUILD EVERYTHING EVERYWHERE

THE GALAXY IS YOURS: 100S OF .NET CONTROLS & TOOLS

COMPONENTONE
ULTIMATE

MEET YOUR DESTINY AT:
COMPONENTONE.COM/GALAXY

ComponentOne

© 2011 ComponentOne LLC. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective holders.

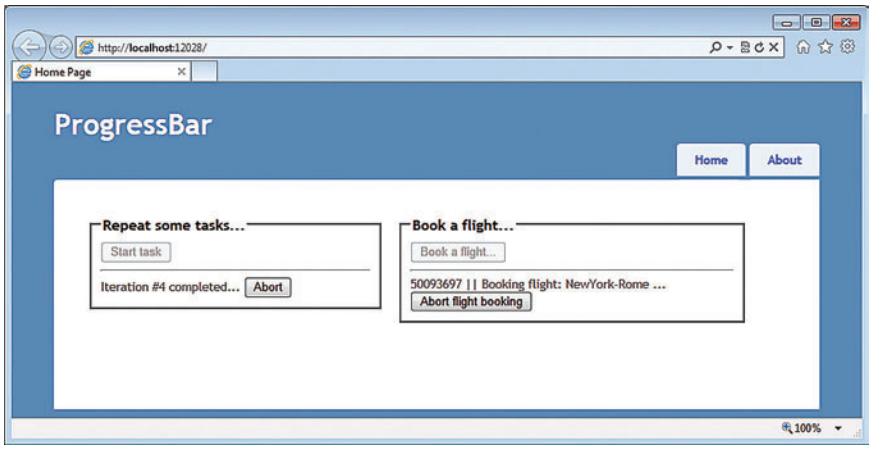


Figure 2 **Cancelable AJAX Operations**

The xhr variable is the reference to the XmlHttpRequest object used in the call. In jQuery, this reference is returned directly by the $.ajax function. **Figure 1** shows an excerpt from the progress framework (renamed ProgressBar) from last month's code that adds a new abort method.

> The truth is, there is no reliable and automatic way to stop the processing of a request.

As you can see, the new abort method doesn't do much beyond calling abort on the internal XmlHttpRequest object. Aborting an ongoing AJAX call still triggers the complete event on the AJAX manager, however, though not any success or error functions. To detect whether an operation was aborted by the user, you attach a handler for complete and check the status property of the XmlHttpRequest object—it will be 0 if the operation was aborted. The complete handler then performs whatever cleanup operation is required—stopping timers, for example. **Figure 2** shows a nice interface from which users can stop remote operations at will.

Figure 3 **The ProgressBar Super Class**

```
public class ProgressBarController : Controller
{
  protected readonly ProgressManager ProgressManager;
  public ProgressBarController()
  {
    ProgressManager = new ProgressManager();
  }

  public String GetTaskId()
  {
    // Get the header with the task ID.
    var id = Request.Headers[ProgressManager.HeaderNameTaskId];
    return id ?? String.Empty;
  }

  public String Status()
  {
    var taskId = GetTaskId();
    return ProgressManager.GetStatus(taskId);
  }

  public void Abort()
  {
    var taskId = GetTaskId();
    ProgressManager.RequestTermination(taskId);
  }
}
```

Notifying the Server

The nice UI in **Figure 2** doesn't necessarily guarantee that the server-side operation has been stopped as the user requested and as the application's feedback seems to prove. Calling abort on XmlHttpRequest simply closes the socket that connects the browser to the server. Put another way, by calling abort on XmlHttpRequest, you simply say you're no longer interested in receiving any response the server method might generate. Nothing really guarantees the server received and acknowledged the abort request; more likely, the server will continue to process the request regardless of whether a browser is listening for a response. While this particular aspect may vary on different platforms and servers, there's another aspect to consider that depends strictly on your application. If the request triggered either an asynchronous operation or a long-running operation, can you stop it? The truth is, there is no reliable and automatic way to stop the processing of a request; you have to build your own framework and write your server methods to be interruptible. Let's extend the progress framework, then.

Figure 4 **A Monitorable Controller Method Using the Progress Framework**

```
public String BookFlight(String from, String to)
{
  var taskId = GetTaskId();

  // Book first leg
  ProgressManager.SetCompleted(taskId,
    "Booking flight: {0}-{1} ...", from, to);
  Thread.Sleep(4000);
  if (ProgressManager.ShouldTerminate(taskId))
  {
    // Compensate here
    //
    return String.Format("One flight booked and then canceled");
  }

  // Book return flight
  ProgressManager.SetCompleted(taskId,
    "Booking flight: {0}-{1} ...", to, from));
  Thread.Sleep(4000);
  if (ProgressManager.ShouldTerminate(taskId))
  {
    // Compensate here
    //
    return String.Format("Two flights booked and then canceled");
  }

  // Book return
  ProgressManager.SetCompleted(taskId,
    "Paying for the flight ...", taskId));
  Thread.Sleep(5000);
  if (ProgressManager.ShouldTerminate(taskId))
  {
    // Compensate here
    //
    return String.Format("Payment canceled. No flights booked.");
  }

  // Some return value
  return "Flight booked successfully";
}
```

Kendo UI

THE ART OF WEB DEVELOPMENT

Everything You Need

For JavaScript & HTML5 development

Kendo UI is a complete framework for JavaScript developers. It provides everything you need to build HTML5 and JavaScript sites and apps, including a powerful data source, crazy-fast templating, rich UI widgets, and customizable themes. Don't waste time piecing together a JavaScript framework. Download Kendo UI and start developing amazing JavaScript apps right away.

Download the future of JavaScript development
at **www.kendoui.com** or scan

Figure 5 **Markup for Triggering a Monitorable and Interruptible Action**

```
<fieldset>
  <legend>Book a flight...</legend>
  <input id="buttonStart" type="button" value="Book a flight..." />

  <hr />
  <div id="progressbar_container">
    <span id="progressbar2"></span>
    <input id="buttonAbort" type="button"
      value="Abort flight booking"
      disabled="disabled" />
  </div>
</fieldset>
```

Extending the Progress Framework

The server-side manager component is the part of the framework that controllers work with. Controller methods call methods on the following interface to post messages for the client progress bar and to receive notifications from the UI to stop processing:

```
public interface IProgressManager
{
    void SetCompleted(String taskId, String format, params Object[] args);
    void SetCompleted(String taskId, Int32 percentage);
    void SetCompleted(String taskId, String step);
    String GetStatus(String taskId);
    void RequestTermination(String taskId);
    Boolean ShouldTerminate(String taskId);
}
```

Compared with the code I presented last month, there are a couple of extra methods—RequestTermination, which clients will call to request termination, and ShouldTerminate, which action methods will invoke to see if they're supposed to stop and roll back.

Each progress manager works on top of a data provider that holds the status of pending tasks, each identified with a client-generated ID. The default data provider in the source code uses the ASP.NET cache to store the status of tasks. It creates an entry for each task and stores the entry inside an object of type TaskStatus, like so:

```
public class TaskStatus
{
  public TaskStatus(String status) : this (status, false)
  {
  }
  public TaskStatus(String status, Boolean aborted)
  {
    Aborted = aborted;
    Status = status;
  }

  public String Status { get; set; }
  public Boolean Aborted { get; set; }
}
```

When the controller method calls SetCompleted, it ends up saving a status message for the task in the underlying store. Before proceeding with the next step, the controller method checks whether it has to abort.

Putting It All Together: The Server

Let's see what it takes to create a controller method for a multistep, monitorable and interruptible operation. You start with a sample controller class that inherits from ProgressBarController:

```
public class TaskController : ProgressBarController
{
  public String BookFlight(String from, String to)
  {
    ...
  }
}
```

Figure 6 **JavaScript Code for the Sample View**

```
function buttonStartHandler() {
  updateStatusProgressBar ();

  progressbar = new ProgressBar();
  progressbar.setInterval(600)
             .callback(function (status) {
                          $("#progressbar").text(status); },
                       function (response) {
                          $("#progressbar").text(response);
                          updateStatusProgressBar(); },
                       function () {
                          $("#progressbar").text("");
                          updateStatusProgressBar(); })
             .start("/task/bookflight?from=Rome&to=NewYork",
                    "/task/status",
                    "/task/abort");
}

function buttonAbortHandler() {
    progressbar.abort();
}

function updateStatusProgressBar () {
    $("#buttonStart").toggleDisabled();
    $("#buttonAbort").toggleDisabled();
}
```

The sample method returns a String for simplicity; it can be a partial view or JSON or whatever else you need it to be.

The base class, shown in **Figure 3**, is a simple way to endow the final controller with a bunch of common methods.

Note, in particular, the Status and Abort methods that define a public and standard API for jQuery clients to call to query for the current status and to request termination. By using a base class, you avoid having to rewrite that code over and over again.

Figure 4 shows the pattern for a controller method that needs to be monitored from the client.

> Each progress manager works on top of a data provider that holds the status of pending tasks.

The task ID is generated on the client and transmitted to the server through an HTTP header. The GetTaskId method shields controller developers from having to know these details. The controller method does its work piecemeal and invokes SetCompleted each time it accomplishes a significant part of the work. It indicates the work done using a string, which could be a percentage as well as a status message. Periodically, the controller method checks whether a request for termination has been received. If this is the case, it does whatever is possible to roll back or compensate, and then returns.

Putting It All Together: The Client

On the client side, you need to link the Progress Framework JavaScript API and the jQuery library:

```
<script src="@Url.Content("~/Scripts/progressbar-fx.js")"
        type="text/javascript"></script>
```

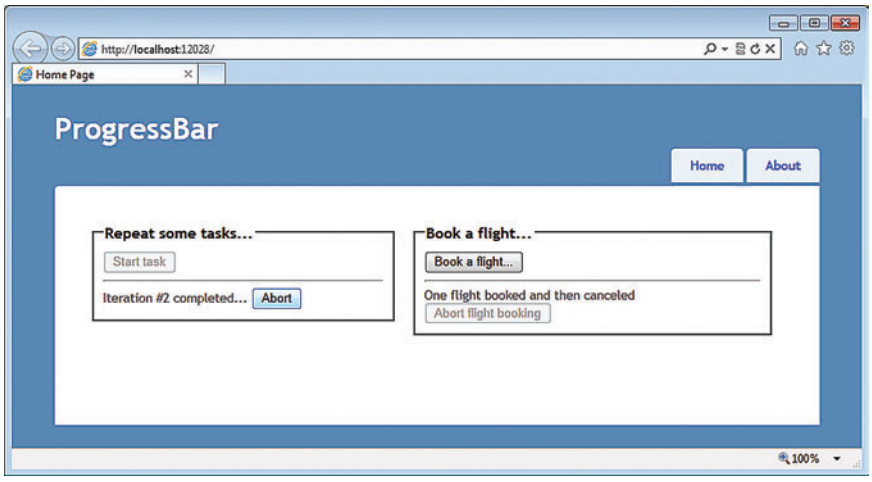


Figure 7 **The Framework in Action**

Each monitorable method will be invoked via AJAX and will therefore be triggered by a client-side event—for example, a button click, created with markup as shown in **Figure 5**.

Click handlers are attached unobtrusively when the page is loaded:

```
<script type="text/javascript">
  var progressbar;

  $(document).ready(function () {
    $("#buttonStart").bind("click", buttonStartHandler);
    $("#buttonAbort").bind("click", buttonAbortHandler);
  });
</script>
```

Figure 6 shows the JavaScript for starting and aborting a remote operation and updating the UI accordingly.

The ProgressBar JavaScript object consists of three main methods. The setInterval method specifies the interval between two successive checks for state updates. The value to pass is in milliseconds. The callback method sets a bunch of callback functions for updating the status and for updating the UI when the operation is successfully completed or when the operation is aborted by the user. And the start method begins the operation. It takes three URLs: the endpoint of the method to run and endpoints for the methods to be called back to catch state updates and to abort the pending operation.

As you can see, the URLs are relative and expressed in the form /controller/method. Of course, you can change the names of the method status and abort to whatever you like—as long as such methods exist as public endpoints. The status and abort methods are guaranteed to exist if you inherit your controller class from ProgressBarController. **Figure 7** shows the sample application in action. Though the UIs in **Figure 2** and **Figure 7** look the same, the underlying code and behavior are really different.

Making Progress

AJAX supplies the tools to poll the server and ask what's going on. You have to build your own framework if you want it to provide monitorable and interruptible methods. It should be noted that in a real example, the action method itself might be calling other asynchronous services. Such code can be complicated. Luckily, writing asynchronous code should get simpler in the upcoming ASP.NET MVC 4. And in my next column, I'll show another approach to implementing and monitoring remote tasks based on a new client-side library that could also make it to the ASP.NET MVC 4 bundle—the SignalR library. For now, get the source code from code.msdn.microsoft.com/mag201201CuttingEdge and let me know your thoughts! ■

Dino Esposito *is the author of "Programming Microsoft ASP.NET MVC3" (Microsoft Press, 2011) and coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. You can follow him on Twitter at twitter.com/despos.*

Thanks to the following technical expert for reviewing this column: *Phil Haack*

The world leader in advanced Microsoft SQL Server Integration Services (SSIS) tasks, components and scripts

5th ANNIVERSARY

SAS® Adapters
Reusable Scripts
USPS Address Parse
Parallel Loop
EDI Source
Table Difference
And More ....

CozyRoc is rare breed among technology companies

Over 100 Reusable Components

CozyRoc
Go to the next level

www.cozyroc.com
sales@cozyroc.com
(919) 249-7421

Develop Once, Deploy Anywhere.

ComponentArt's unique technology targets the widest range of devices with a single code base. Develop once in Visual Studio and XAML, deploy anywhere in a variety of formats.

HTML5

HTML5 is supported by virtually any device: Apple iPad & iPhone, Android Tablets & Phones, Windows Tablets & Phones, Blackberry Playbook & Phones, and any PC or Mac.

XAML/WinRT

Windows 8 features native XAML support and a new WinRT library. Deploy immersive Metro XAML apps through ComponentArt Data Visualization for XAML/WinRT.

Silverlight & WPF

Classic Windows interface continues to be supported through mature .NET development tools: ComponentArt Data Visualization for Silverlight & WPF.

Single code base. Any output format.

Powered By: Microsoft Visual Studio · ComponentArt Data Visualization · ComponentArt Dashboard Server

Sounds unbelievable?

See for yourself at www.ComponentArt.com

ComponentArt
Your Data in a Whole New Light

JULIE LERMAN

Making Do with Absent Foreign Keys

This month I'm writing about an issue I've found myself helping people with frequently of late: problems with related classes defined in Code First and then, in most cases, used in the Model-View-Controller (MVC) framework. The problems developers have been experiencing aren't specific to Code First. They're the result of underlying Entity Framework (EF) behavior and, in fact, common to most object-relational mappers (ORMs). But it seems the problem is surfacing because developers are coming to Code First with certain expectations. MVC is causing them pain because of its highly disconnected nature.

Rather than only showing you the proper code, I'll use this column to help you understand the EF behavior so you can apply this knowledge to the many scenarios you may encounter when designing your classes or writing your data access code with EF APIs.

Code First convention is able to detect and correctly infer various relationships with varying combinations of properties in your classes. In this example, which I'm writing as the leaves are turning spectacular colors on trees near my home in Vermont, I'll use Tree and Leaf as my related classes. For a one-to-many relationship, the simplest way you could describe that in your classes and have Code First recognize your intent is to have a navigation property in the Tree class that represents some type of collection of Leaf types. The Leaf class needs no properties pointing back to Tree. **Figure 1** shows the Tree and Leaf classes.

By convention, Code First will know that a foreign key is required in the database in the Leaf table. It will presume a foreign key field name to be "Tree_TreeId," and with this information provided in the metadata created by Code First at run time, EF will understand how to work out queries and updates using that foreign key. EF leverages this behavior by relying on the same process it uses with "independent associations"—the only type of association we could use prior to the Microsoft .NET Framework 4—which don't require a foreign key property in the dependent class.

This is a nice, clean way to define the classes when you're confident that you have no need to ever navigate from a Leaf back to its Tree in your application. However, without direct access to the foreign key, you'll need to be extra diligent when coding.

Creating New Dependent Types Without Foreign Key or Navigation Properties

Although you can easily use these classes to display a tree and its leaves in your ASP.NET MVC application and edit leaves, developers often encounter issues creating new leaves in a typically architected MVC app. I used the template from the MVCScaffolding NuGet package (mvcscaffolding.codeplex.com) to let Visual Studio automatically build my controllers, views and simple repositories by selecting "MvcScaffolding: Controller with read/write action and views, using repositories." Note that because there's no foreign key property in the Leaf class, the scaffolding templates won't recognize the one-to-many relationship. I made some minor changes to the views and controllers to allow a user to navigate from a tree to its leaves, which you can see in the sample download.

Figure 1 **Related Tree and Leaf Classes**

```
public class Tree
{
  public Tree()
  {
    Leaves = new List<Leaf>();
  }

  public int TreeId { get; set; }
  public string Type { get; set; }
  public double Lat { get; set; }
  public double Long { get; set; }
  public string Notes { get; set; }
  public ICollection<Leaf> Leaves { get; set; }
}
public class Leaf
{
  public int LeafId { get; set; }
  public DateTime FellFromTreeDate { get; set; }
  public string FellFromTreeColor { get; set; }
}
```

The Create postback action for Leaf takes the Leaf returned from the Create view and tells the repository to add it and then save it, as shown in **Figure 2**.

The repository takes the leaf, checks to see if it's new and if so, adds it to the context instance that was created as a result of the postback:

```
public void InsertOrUpdate(Leaf leaf,int treeId)
{
  if (leaf.LeafId == default(int)) {
    // New entity
    context.Leaves.Add(leaf);
  } else {
    // Existing entity
    context.Entry(leaf).State = EntityState.Modified;
  }
}
```

When Save is called, EF creates an Insert command, which adds the new leaf to the database:

```
exec sp_executesql N'insert [dbo].[Leaves]([FellFromTreeDate], [FellFromTreeColor],
[Tree_TreeId]) values (@0, @1, null)
select [LeafId]
from [dbo].[Leaves]
where @@ROWCOUNT > 0 and [LeafId] = scope_identity()',
N'@0 datetime2(7),@1 nvarchar(max) ',
@0='2011-10-11 00:00:00',@1=N'Pale Yellow'
```

Code download available at code.msdn.microsoft.com/mag201201DataPoints.

The best ideas evolve.

Great ideas don't just happen.
They evolve. Your own development
teams think and work fast.

Don't miss a breakthrough.
Version *everything* with Perforce.

Software and firmware. Digital assets
and games. Websites and documents.
More than 5,500 organizations and
400,000 users trust Perforce SCM
to version work enterprise-wide.

Try it now. Download the free 20-user,
non-expiring Perforce Server from
perforce.com/trial

Or request an evaluation license
for any number of users.

PERFORCE
Version everything.

Figure 2 **Adding and Saving Leaf to the Repository**

```
[HttpPost]
public ActionResult Create(Leaf leaf)
{
  if (ModelState.IsValid)
  {
    leafRepository.InsertOrUpdate(leaf);
    leafRepository.Save();
    return RedirectToAction("Index",
      new { treeId = Request.Form["TreeId"] });
  }
  else
  {
    return View();
  }
}
```

Notice the values passed in on the second line of the command: @0 (for the date); @1 (for the modified color); and null. The null value is destined for the Tree_TreeId field. Remember that the nice, clean Leaf class has no foreign key property to represent the TreeId, so there's no way to pass that value in when creating a standalone leaf.

When the dependent type (in this case, Leaf) has no knowledge of its principal type (Tree), there's only one way to do an insert: The Leaf instance and the Tree instance must be added to the context together as part of the same graph. This will provide EF with all the information it needs to work out the correct value to insert into the database foreign key (for example, Tree_TreeId). But in this case, where you're working only with the Leaf, there's no information in memory for EF to determine the value of the Tree's key property.

If you had a foreign key property in the Leaf class, life would be so much simpler. It's not too difficult to keep a single value at hand when moving between controllers and views. In fact, if you look at the Create action in **Figure 2**, you can see that the method has access to the value of the TreeId for which the Leaf is being created.

There are a number of ways to pass data around in MVC applications. I chose the simplest for this demo: stuffing the TreeId into the MVC ViewBag and leveraging Html.Hidden fields where necessary. This makes the value available as one of the view's Request.Form items.

Because I have access to the TreeId, I'm able to build the Tree/Leaf graph that will provide the TreeId for the Insert command. A quick modification to the repository class lets the InsertOrUpdate method accept that TreeId variable from the view and retrieves the Tree instance from the database using the DbSet.Find method. Here's the affected part of the method:

```
public void InsertOrUpdate(Leaf leaf,int treeId)
{
  if (leaf.LeafId == default(int)) {
    var tree=context.Trees.Find(treeId);
    tree.Leaves.Add(leaf);
  }
...
```

The context is now tracking the tree and is aware that I'm adding the leaf to the tree. This time, when context.SaveChanges is called, EF is able to navigate from the Leaf to the Tree to discover the key value and use it in the Insert command.

Figure 3 shows the modified controller code using the new version of InsertOrUpdate.

Figure 3 **The New Version of InsertOrUpdate**

```
[HttpPost]
public ActionResult Create(Leaf leaf)
{
  if (ModelState.IsValid)
  {
    var _treeId = Request.Form["TreeId"] as int;
    leafRepository.InsertOrUpdate(leaf, _treeId);
    leafRepository.Save();
    return RedirectToAction("Index", new { treeId = _treeId });
  }
  else
  {
    return View();
  }
}
```

With these changes, the insert method finally has the value for the foreign key, which you can see in the parameter called "@2":

```
exec sp_executesql N'insert [dbo].[Leaves]([FellFromTreeDate],
[FellFromTreeColor], [Tree_TreeId])
values (@0, @1, @2)
select [LeafId]
from [dbo].[Leaves]
where @@ROWCOUNT > 0 and [LeafId] = scope_identity()',
N'@0 datetime2(7),@1 nvarchar(max) ,
@2 int',@0='2011-10-12 00:00:00',@1=N'Orange-Red',@2=1
```

In the end, this workaround forces me to make another trip to the database. This is the price I'll choose to pay in this scenario where I don't want the foreign key property in my dependent class.

Problems with Updates When There's No Foreign Key

There are other ways you can paint yourself into a corner when you're bound and determined not to have foreign key properties in your classes. Here's another example.

I'll add a new domain class named TreePhoto. Because I don't want to navigate from this class back to Tree, there's no navigation property, and again, I'm following the pattern where I don't use a foreign key property:

```
[Table("TreePhotos")]
public class TreePhoto
{
  public int Id { get; set; }
  public Byte[] Photo { get; set; }
  public string Caption { get; set; }
}
```

The Tree class provides the only connection between the two classes, and I specify that every Tree must have a Photo. Here's the new property that I added to the Tree class:

```
[Required]
public TreePhoto Photo { get; set; }
```

This does leave the possibility of orphaned photos, but I use this example because I've seen it a number of times—along with pleas for help—so I wanted to address it.

Once again, Code First convention determined that a foreign key property would be needed in the database and created one, Photo_Id, on my behalf. Notice that it's non-nullable. That's because the Leaf.Photo property is required (see **Figure 4**).

Your app might let you create trees before the photos have been taken, but the tree still needs that Photo property to be populated.

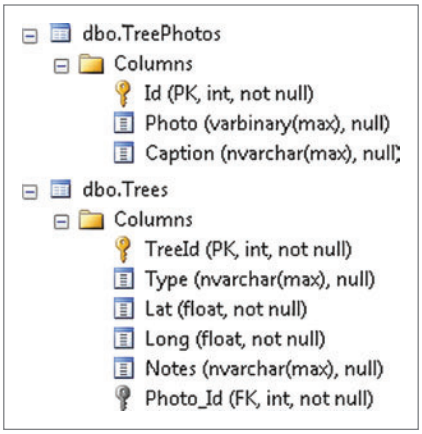
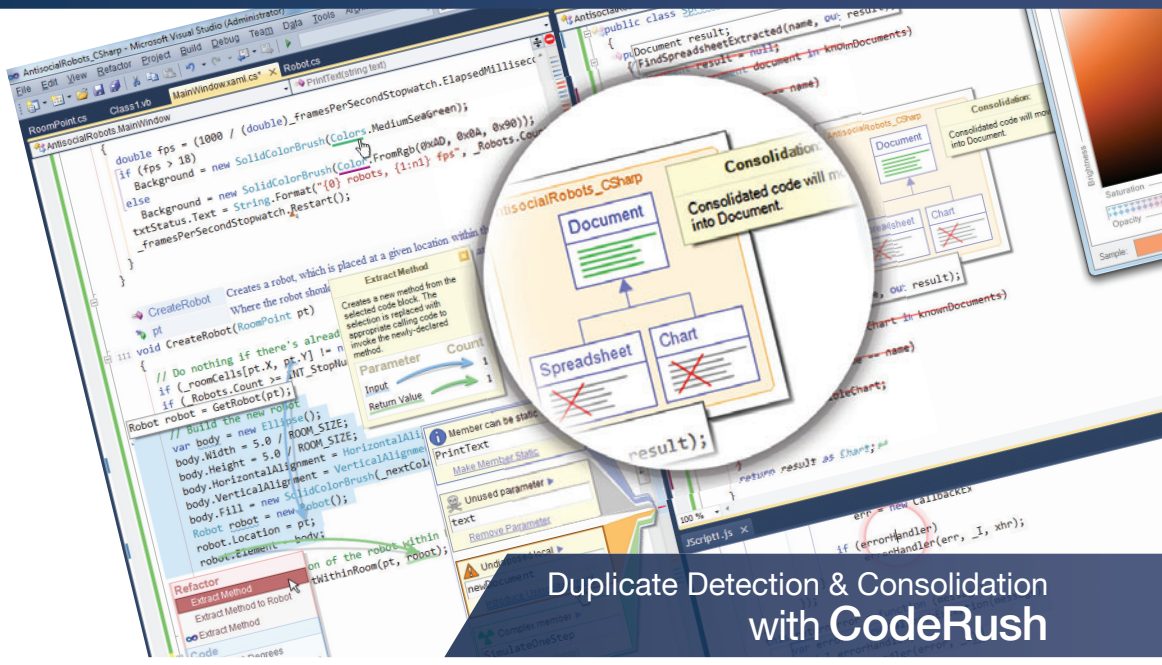


Figure 4 **Using Code First Convention, Tree Gets a Non-Nullable Foreign Key to TreePhotos**

Microsoft
GOLD CERTIFIED
Partner



DXv2 is here

Efficient. Fast. Elegant. Powerful.

CodeRush helps you create and maintain source code with optimum efficiency. Consume-first declaration, powerful templates, smart selection tools, intelligent code analysis, innovative navigation and an unrivalled collection of refactorings all work together to increase your productivity dramatically. See duplicate code throughout your solution while you work and increase the quality of your source code by consolidating duplicates into a single location.

Download the 30-day free trial to experience the next generation of developer productivity tools at **www.DevExpress.com**

Copyright © 1998-2011 Developer Express Inc. ALL RIGHTS RESERVED. All trademarks or registered trademarks are property of their respective owners.

www.DevExpress.com

DevExpress

I'll add logic into the Tree repository's InsertOrUpdate method to create a default, empty Photo for new Trees when one isn't supplied:

```
public void InsertOrUpdate(Tree tree)
{
  if (tree.TreeId == default(int)) {
    if (tree.Photo == null)
    {
      tree.Photo = new TreePhoto { Photo = new Byte[] { 0 },
                                   Caption = "No Photo Yet" };
    }
    context.Trees.Add(tree);
}
...
```

The bigger problem I want to focus on here is how this issue affects updates. Imagine you have a Tree and its required Photo already stored in the database. You want to be able to edit a Tree and have no need to interact with the Photo. You'll retrieve the Tree, perhaps with code such as "context.Trees.Find(someId)." When it's time to save, you'll get a validation error because Tree requires a Photo. But Tree has a photo! It's in the database! What's going on?

Here's the problem: When you first execute a query to retrieve the table, ignoring the related Photo, only the scalar values of the Tree will be returned from the database and Photo will be null (see **Figure 5**).

Both the MVC Model Binder and EF have the ability to validate the Required annotation. When it's time to save the edited Tree, its Photo will still be null. If you're letting MVC perform its ModelState.IsValid check in the controller code, it will recognize that Photo is missing. IsValid will be false and the controller won't even bother calling the repository. In my app, I've removed the Model Binder validation so I can let my repository code be responsible for any server-side validation. When the repository calls SaveChanges, EF validation will detect the missing Photo and throw an exception. But in the repository, we have an opportunity to handle the problem.

If the Tree class had a foreign key property—for example, int PhotoId—that was required (allowing you to remove the requirement on the Photo navigation property), the foreign key value from the database would've been used to populate the PhotoId property of the Tree instance. The tree would be valid, and SaveChanges would be able to send the Update command to the database. In other words, if there were a foreign key property, the Tree would have been valid even without the Photo instance.

But without the foreign key, you'll again need some mechanism for providing the Photo before saving changes. If you have your Code First classes and context set up to perform lazy loading, any mention of Photo in your code will cause EF to load the instance from the database. I'm still somewhat old-fashioned when it comes to lazy loading, so my personal choice would probably be to perform an explicit load from the database. The new line of code (the last line in the following example, where I'm calling Load) uses the DbContext method for loading related data:

```
public void InsertOrUpdate(Tree tree)
{
  if (tree.TreeId == default(int)) {
  ...
  } else {
    context.Entry(tree).State = EntityState.Modified;
    context.Entry(tree).Reference(t => t.Photo).Load();
  }
}
```

This makes EF happy. Tree will validate because Photo is there, and EF will send an Update to the database for the modified Tree. The key here is that you need to ensure the Photo isn't null; I've shown you one way to satisfy that constraint.

A Point of Comparison

If the Tree class simply had a PhotoId property, none of this would be necessary. A direct effect of the PhotoId int property is that the Photo property no longer needs the Required annotation. As a value type, it must always have a value, satisfying the requirement that a Tree must have a Photo even if it isn't represented as an instance. As long as there's a value in PhotoId, the requirement will be satisfied, so the following code works:

```
public class Tree
{
  // ... Other properties
  public int PhotoId { get; set; }
  public TreePhoto Photo { get; set; }
}
```

When the controller's Edit method retrieves a Tree from the database, the PhotoId scalar property will be filled. As long as you force MVC (or whatever application framework you're using) to round-trip that value, when it's time to update the Tree, EF will be unconcerned about the null Photo property.

Easier, but Not Magic

Although the EF team has provided more API logic to help with disconnected scenarios, it's still your job to understand how EF works and what its expectations are when you're moving data around. Yes, coding is much simpler if you include foreign keys in your classes, but they're your classes and you're the best judge of what should and shouldn't be in them. Nevertheless, if your code was my responsibility, I would surely force you to convince me that your reasons for excluding foreign key properties outweighed the benefits of including them. EF will do some of the work for you if the foreign keys are there. But if they're absent, as long as you understand what EF expects and how to satisfy those expectations, you should be able to get your disconnected applications to behave the way you want. ■

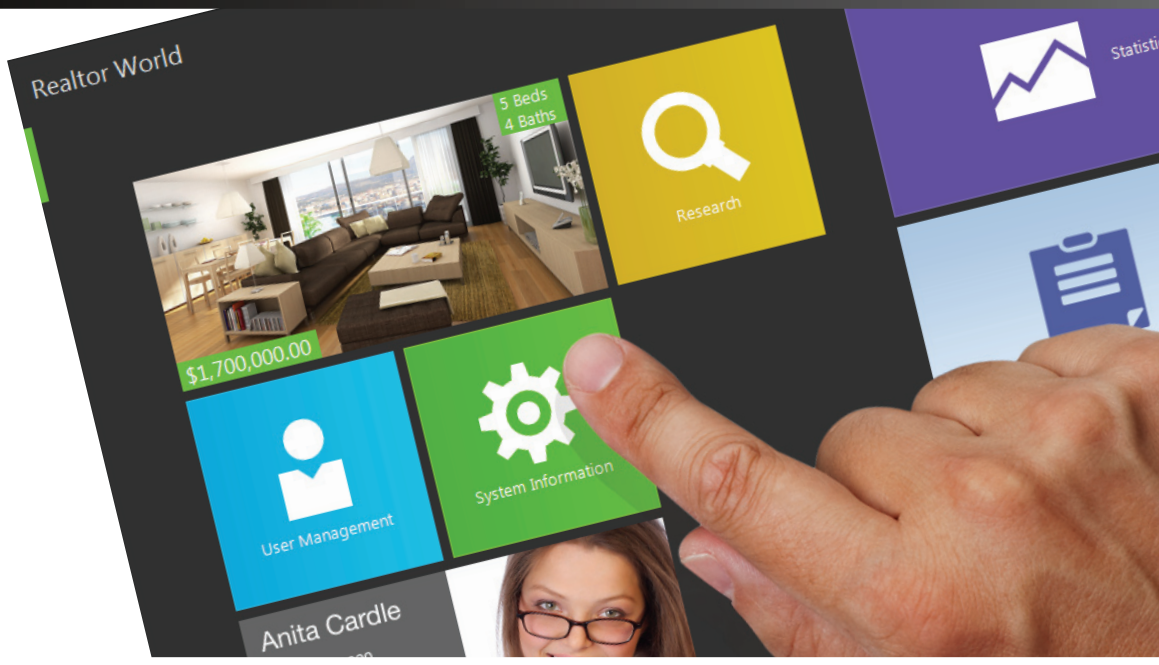
Name	Value
tree	{TreeDomainClasses.Tree}
Lat	44.258
Leaves	Count = 0
Long	-72.956
Notes	"Oh such a lovely tree"
Photo	null
TreeId	1
Type	"Sugar Maple"

Figure 5 **A Tree Instance Retrieved from the Database Without Its Photo**

Julie Lerman *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other Microsoft .NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (2010) and "Programming Entity Framework: Code First" (2011), both from O'Reilly Media. Follow her on Twitter at twitter.com/julielerman.*

Thanks to the following technical experts for reviewing this article: *Jeff Derstadt and Rick Strahl*

Microsoft
GOLD CERTIFIED
Partner



DXv2 is here

Touch at your fingertips.

Bring your software to life with intelligent touch-based applications. Use your existing development skills to tap into the growing demand for stunning tablet & touch-enabled apps across all platforms, including WinForms, WPF and ASP.NET. Build for today as you begin to re-imagine business applications for the Windows 8 Metro design aesthetic. DXv2 delivers the gestures, themes, and controls to put Touch within your reach, right now.

DXv2 is the next generation of tools that can take your applications to a whole new level. Your users are ready—what will you build for them? Download your free 30-day trial at **www.DevExpress.com**

Copyright © 1998-2011 Developer Express Inc. ALL RIGHTS RESERVED. All trademarks or registered trademarks are property of their respective owners.

www.DevExpress.com

DevExpress

JOSEPH FULTZ

Windows Azure Caching Strategies

My two-step with caching started back during the dot-com boom. Sure, I had to cache bits of data at the client or in memory here and there to make things easier or faster for applications I had built up until that point. But it wasn't until the Internet—and in particular, Internet commerce—exploded that my thinking really evolved when it came to the caching strategies I was employing in my applications, both Web and desktop alike.

In this column, I'll map various Windows Azure caching capabilities to caching strategies for output, in-memory data and file resources, and I'll attempt to balance the desire for fresh data versus the desire for the best performance. Finally, I'll cover a little bit of indirection as a means to intelligent caching.

Resource Caching

When referring to resource caching, I'm referring to anything serialized into a file format that's consumed at the endpoint. This includes everything from serialized objects (for example, XML and JSON) to images and videos. You can try using headers and meta tags to influence the cache behavior of the browser, but too often the suggestions won't be properly honored, and it's almost a foregone conclusion that service interfaces will ignore the headers. So, giving up the hope that we can successfully cache slowly changing resource content at the Web client—at least as a guarantee of performance and behavior under load—we have to move back a step. However, instead of moving it back to the Web server, for most resources we can use a content delivery network.

Thinking about the path back from the client, there's an opportunity between the front-end Web servers and the client where a waypoint of sorts can be leveraged, especially across broad geographies, to put the content closer to the consumers. The content is not only cached at those points, but more important, it's closer to the final consumers. The servers used for distribution are known collectively as a content delivery/distribution network. In the early days of the Internet explosion, the idea and implementations of distributed resource caching for the Web were fairly new, and companies such as Akami Technologies found a great opportunity in selling services to help Web sites scale. Fast-forward a decade and the strategy is more important than ever in a world where the Web brings us together while we remain physically apart. For Windows Azure, Microsoft provides the Windows Azure Content Delivery Network (CDN). Although a valid strategy for caching content and moving it closer to the consumer, the reality is that CDN is more typically used by Web sites that have one or both conditions of high-scale and large quantities, or sizes of resources, to serve. A good post about using the Windows Azure CDN can be found on Steve Marx's blog (bit.ly/fvapd7)—he works on the Windows Azure team.

In most cases when deploying a Web site, it seems fairly obvious that the files need to be placed on the servers for the site. In a Windows Azure Web Role, the site contents get deployed in the package—so, check, I'm done. Wait, the latest images from marketing didn't get pushed with the package; time to redeploy. Updating that content currently—realistically—means redeploying the package. Sure, it can be deployed to stage and switched, but that won't be without delay or a possible hiccup for the user.

A straightforward way to provide an updatable front-end Web cache of content is to store most of the content in Windows Azure Storage and point all of the URIs to the Windows Azure Storage containers. However, for various reasons, it might be preferable to keep the content with the Web Roles. One way to ensure that the Web Role content can be refreshed or that new content can be added is to keep files in Windows Azure Storage and move them to a Local Resource storage container on the Web Roles as needed. There are a couple of variations on that theme available, and I discussed one in a blog post from March 2010 (bit.ly/u08DkV).

In-Memory Caching

While the previous caching discussion really focused on the movement of file-based resources, I'll focus next on all the data and dynamically rendered content of the site. I've done tons of performance testing and optimization focused on the performance of

Figure 1 **Add Content by Cache API**

Add to Cache AppFabric Cache

```
DataCacheFactory cacheFactory=
  new DataCacheFactory(configuration);

DataCache appFabCache  =
  cacheFactory.GetDefaultCache();

string value =
  "This string is to be cached locally.";

appFabCache.Put("SharedCacheString", value);
```

Add to Cache System.Web.Caching

```
System.Web.Caching.Cache LocalCache =
  new System.Web.Caching.Cache();

string value =
  "This string is to be cached locally.";
LocalCache.Insert("localCacheString", value);
```

the site and the database behind it. Without exception, having a solid caching plan and implementation that covers output caching (rendered HTML that doesn't have to be rendered again and can just be sent to the client) and data (usually cache-aside style) will get you very far in improving both scale and performance—assuming the database implementation isn't inherently broken.

The heavy lifting in implementing a caching strategy within a site is in determining what gets cached and how frequently it's refreshed versus what remains dynamically rendered on each request. Beyond the standard capabilities provided by the Microsoft .NET Framework for output cache and System.Web.Caching, Windows Azure provides a distributed cache named Windows Azure AppFabric Cache (AppFabric Cache).

Distributed Cache

A distributed cache helps solve several problems. For example, although caching is always recommended for site performance, using session state is typically contraindicated even though it provides a contextual cache. The reason is that getting session state requires that a client is tied to a server, which negatively affects scalability, or that it's synchronized across the servers in a farm, which is generally acknowledged—for good reason—to have issues and limitations. The session-state problem is solved by using a capable and stable distributed cache to back it up. This allows the servers to have the data without continually reaching off the box to get it, and at the same time provides a mechanism to write to the data and have it seamlessly propagated across the cache clients. This gives the developer a rich contextual cache while maintaining the scale qualities of a Web farm.

The best news about AppFabric Cache is that you can use it without doing much more than changing some configuration settings when it comes to session state, and it has an easy-to-use API for programmatic use. Take a look at Karandeep Anand's and Wade Wegner's article in the April 2011 issue for some good details on using the cache (msdn.microsoft.com/magazine/gg983488).

Unfortunately, if you're working with an existing site that directly calls System.Web.Caching in the code, weaving AppFabric Cache in will be a bit more work. There are two reasons for this:

1. The difference in APIs (see **Figure 1**)
2. The strategy of what to cache and where

Figure 1 illustrates clearly that when you look at even the basic elements of the APIs, there's definitely a difference. Creating a layer of indirection to broker the calls will help with the agility of the code in your application. Obviously, some work will be required to easily provide the ability to use the advanced features of the three cache types, but the benefits will outweigh the effort to implement the functionality needed.

Although the distributed cache does solve some generally difficult problems, it shouldn't be used as the snake oil that cures all or it will likely have about the same efficacy as snake oil. First, depending on how things are balanced and the data that goes into the cache, it's possible that more off-machine fetches will be required to get data into the local cache client, which would negatively affect performance. More important is the cost of deployment. As of this writing, the cost of 4GB of AppFabric shared cache is $325 per month. Although this isn't a large amount of money by itself, and 4GB does seem like a good bit of cache space, on a high-traffic site, especially one backing session state with AppFabric Cache and a lot of rich targeted content, it would be easy to fill multiple caches of that size. Consider product catalogs that have price differences based on customer tiers or custom contract pricing.

Figure 2 **Declaring an Enum to Implement a Custom Attribute**

```
public enum CacheLocationEnum
{
  None=0,
  Local=1,
  Shared=2
}

public class CacheLocation:Attribute
{
  private CacheLocationEnum _location = CacheLocationEnum.None;

  public CacheLocation(CacheLocationEnum location)
  {
    _location = location;
  }

  public CacheLocationEnum Location { get { return _location; } }
}
```

Cache-Aside Indirection

Like many things in the technology industry—and I would guess many others—design is some mix of the ideal technical implementations modified by fiscal reality. Thus, even when you're just using Windows Server 2008 R2 AppFabric Caching, there are reasons to still use the local caching provided by System.Web.Caching. At a first pass of indirection, I might have wrapped the calls to each of the caching libraries and provided a function for each, such as AddtoLocalCache(key, object) and AddtoSharedCache(key, object). However, that means each time a cache operation is needed, the developer makes a rather opaque and personal decision on where the caching should happen. Such logic breaks down quickly under maintenance and on larger teams and will inevitably lead to unforeseen errors because the developer could choose to add an object to an inappropriate cache or add to one cache and accidentally fetch from another. Thus a lot of extra data fetching would be needed because data won't be in the cache or will be in the wrong cache when it's fetched. This leads to scenarios such as noticing unexpectedly poor performance only to find on examination that the add operations were done in one cache and the get operations were inexplicably done in the other for no better reason than that the developer forgot or mistyped. Moreover, when planning a system properly, those data types (entities) will be identified ahead of time and with that definition should also be the ideas of where each entity is used, consistency requirements (especially across load-balanced servers) and how fresh it must be. So, it follows that decisions about where to cache (shared or not) and expiry could be made ahead of time and made part of the declaration.

As I mentioned previously, there should be a plan for caching. Too many times it's haphazardly added to the end of a project, but it should be given the same weight of consideration and design as any other aspect of the application. This is especially important when dealing with the cloud, because decisions that aren't well considered often lead to extra cost in addition to the app behavior

Figure 3 **Adding Content to the Cache**

```
public static bool AddToCache<T> (string key, T newItem)
{
  bool retval = false;

  Type curType = newItem.GetType();
  CacheLocation cacheLocationAttribute =
    (CacheLocation) System.Attribute.GetCustomAttribute(typeof(T),
    typeof(CacheLocation));

  switch (cacheLocationAttribute.Location)
  {
    case CacheLocationEnum.None:
      break;
    case CacheLocationEnum.Local:
      retval = AddToLocalCache(key, newItem);
      break;
    case CacheLocationEnum.Shared:
      retval = AddToSharedCache(key, newItem);
      break;
  }

  return retval;
}
```

deficits. When considering the types of data that should be cached, one option is to identify the entities (data types) involved and their lifecycle within the application and user session. Looking at it this way reveals quickly that it would be nice if the entity itself could just intelligently cache based on its type. Luckily, this is an easy task with some assistance from a custom Attribute.

I'm skipping the setup for either cache because the previously referenced material covers that well enough. For my caching library, I've simply created a static class with static methods for my sample. In other implementations, there are good reasons to do this with instance objects, but for the simplicity of this example, I'm making it static.

> My names for types make it obvious where the data will be cached.

I'll declare an enum to indicate location and class that inherits Attribute to implement my custom attribute, as shown in **Figure 2**.

Passing the location in the constructor makes it easy to use in the code later, but I'll also provide a read-only method to fetch the value because I'll need this for a case statement. Within my CacheManager library, I've created a couple of private methods for adding to the two caches:

```
private static bool AddToLocalCache(string key, object newItem)
{...}
private static bool AddToSharedCache(string key, object newItem)
{...}
```

For a real implementation, I'll likely need some other info (for example, cache name, dependencies, expiry and so on), but for now this will do. The main public function for adding content to the cache is a template method, making it easy for me to determine cache from the type, as shown in **Figure 3**.

I'll simply use the passed-in type to get the custom attribute and ask for my custom attribute type via the GetCustomAttribute(type, type) method. Once I have that, it's a simple call to the read-only property and a case statement and I've successfully routed the call to the appropriate cache provider. To ensure that it works properly, I need to adorn my class declarations appropriately:

```
[CacheLocation(CacheLocationEnum.Local)]
public class WebSiteData
{
  public int IntegerValue { get; set; }
  public string StringValue { get; set; }
}

[CacheLocation(CacheLocationEnum.Shared)]
public class WebSiteSharedData
{
  public int IntegerValue { get; set; }
  public string StringValue { get; set; }
}
```

With all of my application infrastructure set up, it's ready for me to consume within the application code. I crack open the default.aspx.cs file to create the sample calls and add code to create the types, assign some values and add them to the cache:

```
WebSiteData data = new WebSiteData();
data.IntegerValue = 10;
data.StringValue = "ten";

WebSiteSharedData sharedData = new WebSiteSharedData();
sharedData.IntegerValue = 50;
sharedData.StringValue = "fifty";

CachingLibrary.CacheManager.AddToCache<WebSiteData>("localData", data);
CachingLibrary.CacheManager.AddToCache<WebSiteSharedData>(
  "sharedData", sharedData);
```

My names for types make it obvious where the data will be cached. However, I could change the type names and it would be less obvious with the caching controlled by inspection of the custom Attribute. Using this pattern will hide from the page developer the details of where the data gets cached and other details related to the cache item configuration. Thus, those decisions are left to the part of the team that's creating the data dictionaries and prescribing the overall lifecycle of said data. Note the type being passed into the calls to AddToCache<t>(string, t). Implementing the rest of the methods for the CacheManager class (that is, GetFromCache) would take on the same pattern as used here for the AddToCache method.

Balancing Cost with Performance and Scale

Windows Azure provides the necessary software infrastructure to help you with any aspect of your implementation, including caching and whether the caching is for resources such as those distributed via CDN or data that might be kept in the AppFabric Cache. The key to a great design and subsequently great implementation is to balance cost with performance and scale. One last note: If you're working on a new application right now and are planning on building caching into it, go ahead and put that layer of indirection in now. It's a little extra work, but as new features such as AppFabric Caching come online, this practice will make it easier to thoughtfully and effectively incorporate the new features into your application. ■

JOSEPH FULTZ *is a software architect at Hewlett-Packard Co., working as part of the HP.com Global IT group. Previously he was a software architect for Microsoft, working with its top-tier enterprise and ISV customers defining architecture and designing solutions.*

THANKS to the following technical expert for reviewing this article: *Wade Wegner*

Hey Developers
Are you working with FILES?

Aspose provides premium file APIs for most business-centric formats.
.NET, Java, SharePoint, SQL Reporting & JasperReports

Aspose.Words

DOC, DOCX, RTF, HTML, PDF, XPS & other document formats.

Aspose.Cells

XLS, XLSX, XLSM, XLTX, CSV SpreadsheetML & image formats.

Aspose.Slides

PPT, PDF, PPS, POTX, PPTX & other formats.

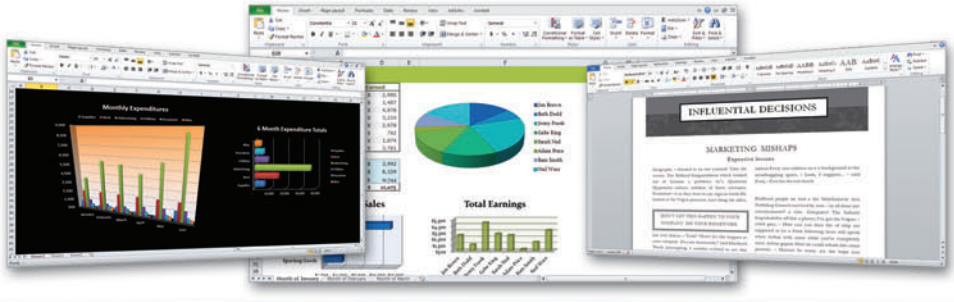
Aspose.Pdf Aspose.BarCode Aspose.Tasks
Aspose.Email Aspose.Diagram Aspose.OCR

Features:

- Create
- Edit
- Convert
- Import
- Export
- Render
- Save
- Print

When you buy Aspose.Total, you will receive our complete suite and any future products.

100% Standalone - No Office Automation



ASPOSE
Your File Format Experts

Get your FREE evaluation copy at **http://www.aspose.com**.

US Sales: 1.888.277.6734 · sales@aspose.com · EU Sales: +44 (0)800 098 8425 · sales.europe@aspose.com

Enterprise Sales – enterprise.sales@aspose.com

Your First Windows Phone Application

Jesse Liberty

The trick in writing a first Windows Phone application is to build something interesting enough to be meaningful, yet simple enough to actually get underway. Toward that end, I'll walk you through creating a simple utility that I use every day: NoteToMe. The idea is that you can enter a message and send it to yourself by pressing one button, as shown in **Figure 1**.

Note that this article will touch on a number of topics, each of which will be expanded upon at some length in subsequent articles. These topics include:

- Creating the layout of the application
- Storing and retrieving data from Isolated Storage
- Events and event handling
- Creating and running tasks (launchers and choosers)

Before you can begin, you'll need to download the tools from create.msdn.com. If you've already unlocked your phone but not yet upgraded to Windows Phone 7.5 ("Mango"), now is the time to do so; Mango brings hundreds of new features to the Windows Phone OS.

Getting Started

Like many Windows Phone developers, I've come to believe that the best tool for creating Windows Phone applications is a combination of Visual Studio (for code) and Expression Blend (for everything else). Thus, we'll begin by opening Expression Blend and creating a new application named NoteToMe, based on the Windows Phone SDK 7.1.

Let's start by changing the application title. Click on the title, and in the Properties window find the Text property for that control. The Metro design guidelines (the design guidelines for Windows Phone) call for the title to be in all caps, so change the title to NOTE TO ME.

Click on the page title and hit Delete to remove it.

To create the layout, you'll need a small row near the top of the page. Click in the margin to bring up a guideline that helps you visually select where to place the row, as shown in **Figure 2**.

Of course, you can set the row size by hand directly in the XAML:

```
<Grid.RowDefinitions>
  <RowDefinition Height="1*"/>
  <RowDefinition Height="9*"/>
</Grid.RowDefinitions>
```

The asterisk after the value indicates relative sizing—in this case 1:9. That is, the first row will be one-ninth of the size of the second row.

Adding Three Controls to the StackPanel

The top row will have three controls in it, side by side:

- A TextBlock acting as a label
- A TextBlock to hold the e-mail address
- A Button to send the message

This design is shown in **Figure 3**.

You can't put three controls into a single column of a row without putting them inside some sort of organizing container. I'll use a StackPanel whose orientation has been set to horizontal—StackPanels stack on top of each other or next to each other.

To create a StackPanel, click on the tiny white arrow next to the Layout control on the toolbar as shown in **Figure 4**.

Click on the StackPanel to select the control. Now drag a StackPanel into the row and set its vertical and horizontal alignment to stretch and its margins to zero in the Layout window as shown in **Figure 5**.

This article discusses:

- Creating the layout of the application
- Storing and retrieving data from Isolated Storage
- Events and event handling
- Creating and running tasks (launchers and choosers)

Technologies discussed:

Windows Phone, Visual Studio, Expression Blend

Add the TextBlock, setting its font size to 32, and its text to To. Now drag a TextBox onto the StackPanel. (Note the important but subtle difference between a TextBlock, for displaying text, and a TextBox, for text input.) Name this TextBox Address. Finally, add a button to the StackPanel, name it Send and set its Content to Send.

The XAML this produces is shown in **Figure 6**.

Notice the Send button has a Click="Send_Click" property. You create this by clicking on the button, then in the Properties window, you click on the Events button, as shown in **Figure 7**.

This opens all the events for the button. Find the click event and double-click. The button is updated with the event, and you're placed in the code editor (either in Blend or in Visual Studio, depending on how you have Blend set up) for that event handler. For now, you can leave this event handler as is:

```
private void Send_Click( object sender,
RoutedEventArgs e )
{
}
```

Figure 1 **The NoteToMe Interface**

Adding the Message Control

Click on the TextBox control in the toolbar and then drag a TextBox out to fill half of the remaining page (we're leaving the other half for the keyboard, which will appear when it's time to enter something in the TextBox). Set the HorizontalAlignment to Stretch, the VerticalAlignment to Top, the margins to 0. Set the Width to Automatic and the height to 244. You can do all this by eye as you resize the TextBox, or you can draw the TextBox roughly in place and set the properties in the Properties window as shown in **Figure 8**.

Figure 2 **Placing the Top Row of the Layout**

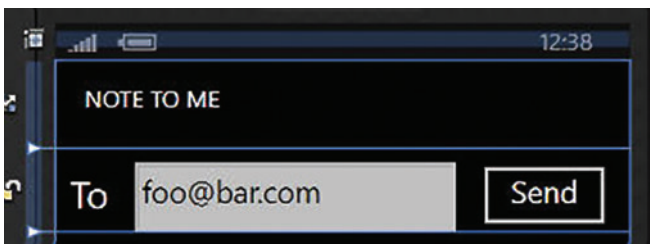


Figure 3 **Three Controls in the Top Row**

Writing the Code

With the controls in place, you're ready to work on the logic of the program. You should see a tab called Projects in the upper left corner. After saving all your changes, click the Projects tab, then right-click on MainPage.xaml.cs and select Edit In Visual Studio, as shown in **Figure 9**.

The Specification

My (self-imposed) specification says you shouldn't have to fill in the To field each time you use the program; the To field should be prepopulated with whatever was in the To field the previous use.

Moreover, when you click Send, a new e-mail message should be prepared for your e-mail program, with all the fields prepopulated so that you can just press Send or, optionally, edit the message and then press Send.

Using Isolated Storage

To preserve the contents of the To field across usages of the application, you need to store the contents somewhere on the phone. This is what Isolated Storage is for: persisting data when the application is closed. As its name implies, Isolated Storage allows your application to store data isolated and protected from data stored by other applications. Using Isolated Storage is fairly straightforward.

First, add the using statement:

```
using System.IO.IsolatedStorage;
```

Declare a member variable of type IsolatedStorageSettings and a constant string to act as a key into the Isolated Storage dictionary so you can store and retrieve the e-mail address:

```
private IsolatedStorageSettings _isoSettings;
const string IsoKey = "EmailAddress";
```

Initialize the _isoSettings member in the constructor:

```
_isoSettings = IsolatedStorageSettings.ApplicationSettings;
```

Storing and Retrieving the E-mail Address

The two tasks related to Isolated Storage are storing the string and retrieving it. Storing it is best done as you leave the page. When you leave any Windows Phone page, the method OnNavigatedFrom is called. You're free to override it, and one good reason to do so is to store data in Isolated Storage, like this:

```
protected override void OnNavigatedFrom(
  System.Windows.Navigation.NavigationEventArgs e )
{
  _isoSettings[IsoKey] = Address.Text;
  base.OnNavigatedFrom( e );
}
```

Now you have the e-mail address stored in the _isoSettings dictionary under the IsoKey key. When you return to the page, you can restore this setting. I do this by calling the private helper method RestoreEmailAddress from the constructor:

```
private void RestoreEmailAddress()
  {
    if (_isoSettings.Contains( IsoKey ))
      Address.Text = _isoSettings[IsoKey].ToString();
  }
```

Notice that I test for the existence of the key in Isolated Storage before trying to restore it—this averts a KeyNotFound exception the first time I run the program. Remember, the first time you run the program you haven't yet stored anything in Isolated Storage.

When the program is first started, there's nothing in the Address field. Once the user puts an e-mail address in the address field, that address is stored in Isolated Storage and restored the next time the program is run. If the user changes the address, that new address is the one restored.

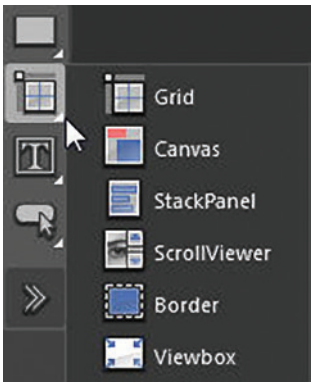


Figure 4 **Adding a StackPanel**

Tasks

Windows Phone 7.5 supports a number of tasks for interacting with built-in phone applications (mail, contact list, the camera and so forth). There are two types of tasks: Launchers and Choosers. Choosers are used to select information and return it to your program (to obtain an e-mail address from the contact list, for example). Launchers are used to launch a program that doesn't return data.

In this case, you have everything you need to send the message, so you can call the e-mail Launcher and supply the required fields. When you call Show on the e-mail Launcher, the e-mail application will be launched with your data, but you won't get any data back (which is fine; you don't need any).

After the e-mail is sent, you're returned to the program in case you want to send another message.

All of the work of creating the Launcher is encapsulated within the click event handler for the Send button. Let's begin by creating an instance of the EmailComposeTask (the Launcher). Fill in the fields and call Show. That's all there is to it:

```
private void Send_Click( object sender, RoutedEventArgs e )
{
  EmailComposeTask emailComposeTask = new EmailComposeTask();
  emailComposeTask.Subject = "Send To Me";
  emailComposeTask.To = Address.Text;
  emailComposeTask.Body = Message.Text;
  Message.Text = String.Empty;
  emailComposeTask.Show();
}
```

When you call Show, the subject, address and body of the message are passed to your e-mail application. If you have more than one e-mail application, you're asked which one you'd like to use. A properly addressed and formatted e-mail message is created, ready for you to send.

The Application Lifecycle

If users could be relied on to never interrupt their use of your application until they sent the message, you'd be done. In reality, though, users will stop right in the middle of composing a message and launch a different application. When they return, they won't be happy if the work they did is gone.

To know how to protect against this, you need to understand a bit about the lifecycle of an application, and how you can preserve state while still supporting one of the most powerful features of Mango: Fast Application Switching.

When your application is launched (say from the Start menu), the Application Launching event is fired. Once the application is started, and every time the user navigates to your page, the OnNavigatedTo method is called, after which your page will be in the Running state. If the user starts a new application, your application receives the Application Deactivated event and is put in the *dormant* state. If the phone runs low on memory, your application may be tombstoned.

From either tombstoned or dormant, your application may be terminated or it may be restored. What we care about right now is what happens when your application is restored.

If your application is dormant, you not only don't have to take any action when it's restored, but you also don't *want* to take any action; the state was preserved when the application was dormant, and it's ready to go.

If your application was tombstoned, though, then you *do* want to restore your page state when the application returns so that it appears to the user that the application was running (or at least dormant) while it was switched away.

You therefore face two tasks:

1. Save state when the page's OnNavigatedFrom method is called.
2. Potentially restore state when the page's OnNavigatedTo method is called—restoring state if the app was tombstoned but *not* if it was dormant.

Saving State When the Page Is Going Away

Because you can't know, when the page receives the OnNavigatedFrom, what state it will be in when it's restored, you must store the state in case it will be needed. This is very easy to do: you use a State dictionary that's very much like the Isolated Storage dictionary in

Figure 5 **Placing the StackPanel**

Figure 6 **Designing the StackPanel with XAML**

```
<StackPanel
  Margin="0"
  Orientation="Horizontal">
  <TextBlock
    Margin="0,8"
    TextWrapping="Wrap"
    Text="To"
    Width="42"
    HorizontalAlignment="Left"
    VerticalAlignment="Center"
    FontSize="32" />
  <TextBox
    x:Name="Address"
    Margin="0,0,0,-7"
    TextWrapping="Wrap"
    Text="foo@bar.com"
    Width="293" />
  <Button
    x:Name="Send"
    Content="Send"
    Margin="0,4,0,0"
    Width="124"
    Click="Send_Click" />
</StackPanel>
```

its syntax, though you want to remember that the State dictionary is not written to permanent storage and is in fact destroyed when you exit the program or turn off the phone.

Let's start by creating a constant string StateKey, which you'll use as the offset into the State dictionary:

```
const string StateKey = "MessageState";
```

In the OnNavigatedFrom method, you'll store the state (in this case, the contents of the MessageBox) into the State dictionary:

```
protected override void OnNavigatedFrom(
  System.Windows.Navigation.NavigationEventArgs e )
{
  _isoSettings[IsoKey] = Address.Text;
  State[StateKey] = Message.Text;
  base.OnNavigatedFrom( e );
}
```

Restoring State When the Page Is Created

When the OnNavigatedTo method is called, you don't want to take any action to restore State if the app was dormant, but you do want to take action if it was tombstoned.

You can distinguish between the dormant or tombstoned state by setting a flag to false, and then setting it to true in the constructor. If the app is dormant, the constructor will not be called; if it was tombstoned, the constructor will be called (because it will be the first time it's constructed), like so:

```
bool isNew = false;
  public MainPage()
  {
    InitializeComponent();
    isNew = true;
```

You can check that flag in OnNavigatedTo:

```
protected override void OnNavigatedTo(
  System.Windows.Navigation.NavigationEventArgs e )
{
  if (isNew)
  {
    if (State.ContainsKey( StateKey ))
    {
      Message.Text = State[StateKey].ToString();
    }
  }
  isNew = false;
  base.OnNavigatedTo( e );
}
```

This test saves the time it would otherwise take to restore the value from the State dictionary. You can test this by first running your program normally (in which case when you switch to another application your program will go dormant) and then forcing the program to be tombstoned. You can force your program to be tombstoned by right-clicking on the project, choosing Properties, choosing the Debug tab and checking the checkbox *Tombstone upon deactivation while debugging.*

When you run with this checked, you'll see a noticeable pause when returning to the page because the state must be restored.

Final Overview

In this brief article, I've shown you how to write your first non-trivial Windows Phone application. I began by creating the application in Expression Blend, where I created a row and used a StackPanel to lay out the controls.

I then switched to Visual Studio to write the logic for the button's event handler, and used Isolated Storage to persist the e-mail address. I used State memory to ensure that the application would restart properly after being tombstoned.

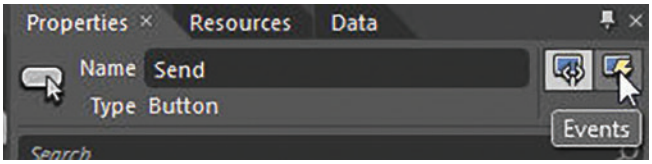


Figure 7 **The Events Button**

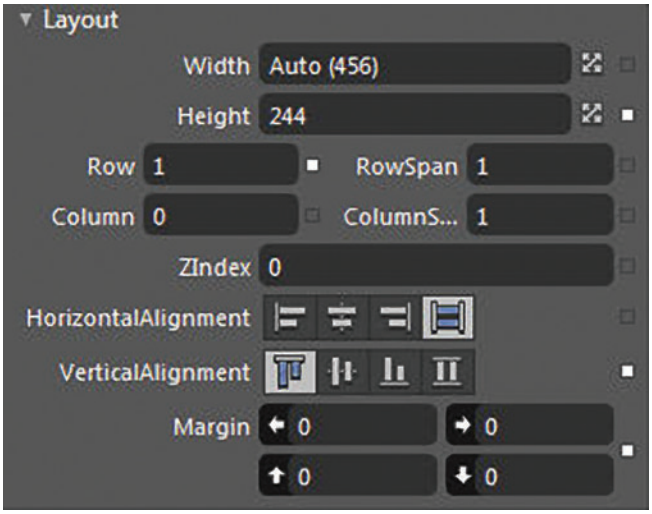


Figure 8 **Adding the TextBox**

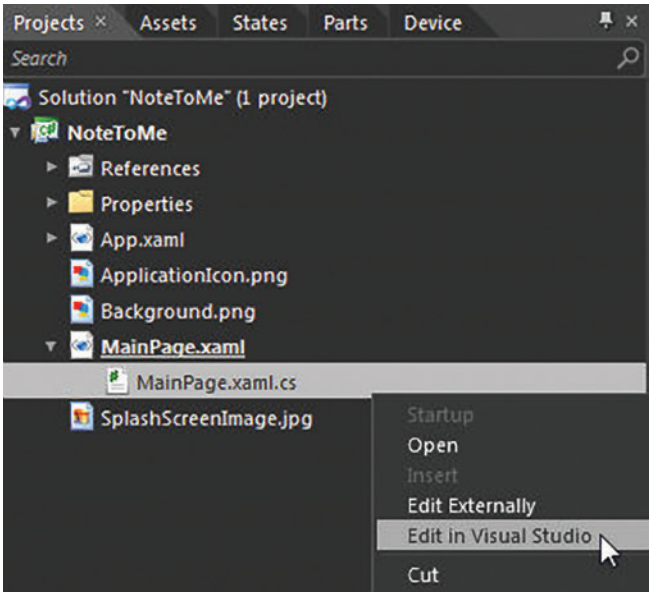


Figure 9 **Getting Ready to Write the Code**

As noted, there's much more to say on each of these topics, and they'll be covered in detail in future articles. ■

JESSE LIBERTY *is a senior developer-community evangelist on the Windows Phone team. Liberty hosts the popular Yet Another Podcast (jesseliberty.com/podcast), and his blog (jesseliberty.com/) is required reading. He's the author of numerous best-selling books, including "Programming Reactive Extensions and LINQ" (Apress, 2011) and "Migrating to Windows Phone," (Apress, 2011). You can follow Liberty on Twitter at twitter.com/JesseLiberty.*

THANKS to the following technical experts for reviewing this article: *Drew Batchelor and Cheryl Simmons*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

WHAT YOU LEARN IN VEGAS WON'T STAY IN VEGAS

Intense Take-Home Training for Developers, Software Architects and Designers

Las Vegas | **March 26-30** | Mirage Resort and Casino

SUPPORTED BY

Microsoft | Microsoft Visual Studio | msdn | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

Las Vegas

March 26-30

Mirage Resort and Casino

Coding, Casinos and More!

If you're looking for unbiased, hard-hitting and practical .NET Developer training, look no further than Visual Studio Live! Las Vegas.

Check out the hot track topics and a sampling of the sessions that will make YOU a more valuable part of your company's development team:

- **Windows 8 / WinRT:** Windows 8 Metro-style Application Contracts and Extensibility
- **Silverlight / WPF:** Top 7 Lessons Learned On My First Big Silverlight Project
- **Web:** MVC For WebForms Developers: Comparing and Contrasting
- **Visual Studio 2010+ / .NET 4.0+:** What's New in the .NET 4.5 BCL
- **Cloud Computing:** Architecture Best Practices on Windows Azure
- **Data Management:** Entity Framework Code First—Beyond the Basics
- **HTML5:** Building Windows 8 Applications with HTML5 and jQuery
- **Windows Phone 7:** XNA Games for Windows Phone
- **Cross Platform Mobile:** Building Mobile Apps with CSLA .NET
- **Full-Day Workshops:** Full Application Lifecycle with TFS and CSLA .NET

Register Before February 1st and Save $400!

Use Promo Code JANAD

vslive.com/lasvegas

Scan this code to register and learn more about what Visual Studio Live! Las Vegas has to offer.

Using Cameras in Windows Phone 7.5

Matt Stroshane

Pictures can communicate with an efficiency and elegance that can't be matched by words alone. You've heard that "a picture is worth a thousand words"; imagine the types of problems you could solve if your Windows Phone application had direct access to a camera. Well, starting with Windows Phone 7.5, you can begin solving those "thousand-word" problems using the on-device cameras.

In this article, I'll introduce the front and back cameras, the camera APIs and the associated manifest capabilities, plus I'll discuss a few different ways you can use a camera in your next Windows Phone 7.5 application. I'll cover:

- Capturing photos: I'll create a very simple photo app.
- Accessing the camera preview buffer: I'll introduce the Camera Grayscale Sample.
- Recording video: I'll review the Video Recorder Sample.

You'll need the Windows Phone SDK 7.1 to create a Windows Phone 7.5 application. The SDK includes code examples that demonstrate each of these scenarios in great detail. For more information, see the Basic Camera Sample, Camera Grayscale Sample, and the Video Recorder Sample on the Code Samples page in the SDK at wpdev.ms/officialsamples.

Note that this article won't cover the camera capture task, which has been available since Windows Phone 7. Though this task is a simple way to acquire photos for your application, it doesn't let you capture photos programmatically or access the camera preview buffer.

A Windows Phone 7.5 device can include up to two cameras, designated as primary and front-facing. The primary camera is on the back of the device and typically offers a higher resolution and more features than the front-facing camera. Neither of these cameras is required on a Windows Phone 7.5 device, so be sure to check for their presence in your code before you create your camera objects. Later on, I'll demonstrate how to use the static IsCameraTypeSupported method for this purpose.

Many of the Windows Phone devices available in the United States include a primary camera with a 5MP or greater sensor, auto-focus and a flash. The front-facing camera is a new feature for Windows Phone 7.5.

For more information about device specifications, see the Buy tab at windowsphone.com.

This article discusses:

- Capturing photos
- Accessing the camera preview buffer
- Recording video

Technologies discussed:

Windows Phone SDK 7.1, Windows Phone 7.5, C#

Capturing Photos

You can use the same classes to access both the primary camera and the front-facing camera. As you'll see, selecting the camera type is simply a matter of specifying a single parameter in the constructor of the PhotoCamera object. From a design perspective, however, you might want to handle interaction with the front-facing

camera differently. For example, you might want to flip images from the front-facing camera to give the user a more natural "mirror-like" experience.

When capturing photos in a Windows Phone 7.5 app, you'll work primarily with the PhotoCamera class from the Microsoft.Devices namespace. This class offers a great deal of control over the camera settings and behavior. For example, you can:

- Activate the camera shutter with the PhotoCamera.CaptureImage method
- Trigger auto focus with the PhotoCamera.Focus method
- Specify picture resolution by setting the PhotoCamera.Resolution property
- Specify the flash settings by setting the PhotoCamera.FlashMode property
- Incorporate the hardware shutter button with events from the static CameraButtons class
- Implement touch focus with the PhotoCamera.FocusAtPoint method

In this article, I'll demonstrate only the first point. For an example that shows how to do all of these, see the Basic Camera Sample from the Windows Phone SDK code samples page.

Note that even when a camera is available, it might not support all of these APIs. The following approaches can help determine what is available:

- Camera: Use the PhotoCamera.IsCameraTypeSupported static method.
- Auto focus: Use the PhotoCamera.IsFocusSupported method.
- Picture resolution settings: Check the PhotoCamera.AvailableResolutions collection.
- Flash settings: Use the PhotoCamera.IsFlashModeSupported method.
- Point-specific focus: Use the PhotoCamera.IsFocusAtPointSupported method.

To give you an idea of how to capture photos in your app, let's walk through a simple app that captures a photo when you touch the viewfinder and then saves it to the Camera Roll folder in the Pictures Hub.

Start with a standard Windows Phone project, using the Windows Phone Application template. You can write Windows Phone 7.5 apps in C# or Visual Basic. This example will use C#.

I'll simplify this example by limiting the app to a landscape-only orientation and using just the primary camera. Managing orientation for the device and two cameras, each pointed in different directions, can become confusing pretty quickly; I recommend testing with a physical device to ensure you achieve the desired behavior. I'll cover orientation in more detail later.

On MainPage.xaml, update the PhoneApplicationPage attributes as follows:

```
SupportedOrientations="Landscape" Orientation="LandscapeLeft"
```

Then, replace the contents of the LayoutRoot grid with Canvas and TextBlock as shown in **Figure 1**.

The XAML in **Figure 1** uses a VideoBrush in a Canvas to display the viewfinder and provides a TextBlock for communicating with the user. The camera sensor has a 4:3 aspect ratio, and the screen aspect ratio is 15:9. If you don't specify a canvas size with the same 4:3 ratio (640x480), the image will appear stretched across the screen.

Figure 1 **Adding a Canvas and a TextBlock**

```
<Canvas x:Name="viewfinderCanvas" Width="640" Height="480" Tap="viewfinder_Tapped">
  <Canvas.Background>
    <VideoBrush x:Name="viewfinderBrush">
      <VideoBrush.RelativeTransform>
        <CompositeTransform
          x:Name="viewfinderTransform"
          CenterX="0.5"
          CenterY="0.5"/>
      </VideoBrush.RelativeTransform>
    </VideoBrush>
  </Canvas.Background>
</Canvas>

<TextBlock Width="626" Height="40"
           HorizontalAlignment="Left"
           Margin="8,428,0,0"
           Name="txtMessage"
           VerticalAlignment="Top"
           FontSize="24"
           FontWeight="ExtraBold"
           Text="Tap the screen to capture a photo."/>
```

In the Canvas element, the Tap attribute specifies the method to call when the user taps the screen—the viewfinder_Tapped method. To display the image stream from the camera preview buffer, a VideoBrush named viewfinderBrush is specified as the background of the canvas. Like a viewfinder from a single-lens reflex (SLR) camera, viewfinderBrush lets you see the camera preview frames. The transform in viewfinderBrush essentially "pins" the viewfinder to the center of the canvas as it's rotated. I'll discuss the code behind this XAML in the following sections. **Figure 2** shows the Simple Photo App UI.

Initializing and Releasing the Camera To capture photos and save them to the Camera Roll folder in the Pictures Hub, you'll need the PhotoCamera and MediaLibrary classes, respectively. Start by adding a reference to the Microsoft.Xna.Framework assembly. You don't need to know XNA programming for this example; you do need types in this assembly, though, to access the media library.

At the top of the MainPage.xaml.cs file, add directives for the camera and media library:

```
using Microsoft.Devices;
using Microsoft.Xna.Framework.Media;
```

In the MainPage class, add the following class-level variables:

```
private int photoCounter = 0;
PhotoCamera cam;
MediaLibrary library = new MediaLibrary();
```

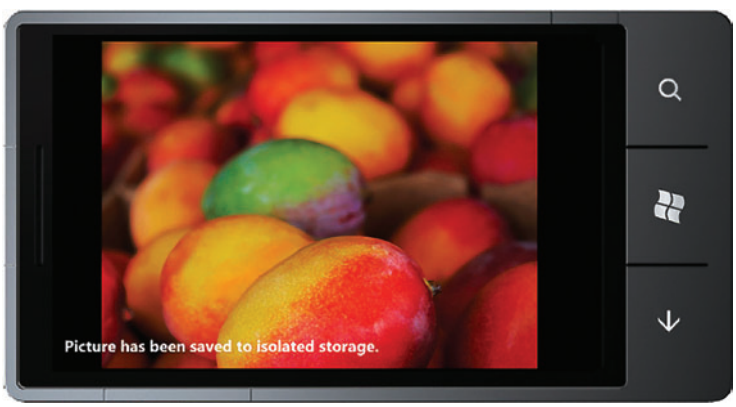


Figure 2 **The Simple Photo App UI**

Figure 3 **The OnNavigatedTo and OnNavigatingFrom Methods**

```
protected override void OnNavigatedTo
  (System.Windows.Navigation.NavigationEventArgs e)
{
  if (PhotoCamera.IsCameraTypeSupported(CameraType.Primary) == true)
  {
    cam = new PhotoCamera(CameraType.Primary);
    cam.CaptureImageAvailable +=
      new EventHandler<Microsoft.Devices.ContentReadyEventArgs>
        (cam_CaptureImageAvailable);
    viewfinderBrush.SetSource(cam);
  }
  else
  {
    txtMessage.Text = "A Camera is not available on this device.";
  }
}


protected override void OnNavigatingFrom
  (System.Windows.Navigation.NavigatingCancelEventArgs e)
{
  if (cam != null)
  {
    cam.Dispose();
  }
}
```

Figure 4 **The viewfinder_Tapped Method**

```
void viewfinder_Tapped(object sender, GestureEventArgs e)
{
  if (cam != null)
  {
    try
    {
      cam.CaptureImage();
    }
    catch (Exception ex)
    {
      this.Dispatcher.BeginInvoke(delegate()
      {
        txtMessage.Text = ex.Message;
      });
    }
  }
}
```

The camera can take a few seconds to initialize. By declaring the PhotoCamera object at the class level, you can create it when you navigate to the page and remove it from memory when you navigate away. We'll use the OnNavigatedTo and OnNavigatingFrom methods for this purpose.

In the OnNavigatedTo method, create the camera object, register for the camera events that will be used, and set the camera preview as the source of the viewfinder, viewfinderBrush. Although common, cameras are optional in Windows Phone 7.5; it's important to check for them before you create the camera object. If the primary camera isn't available, the method writes a message to the user.

Add the methods shown in **Figure 3** to the MainPage class.

When navigating away from the page, use the OnNavigatingFrom method to dispose of the camera object and unregister any camera events. This helps minimize power consumption, expedite shutdown and release memory.

Capturing a Photo As shown in the XAML, when the user taps on the viewfinder, the viewfinder_Tapped method is called. This method initiates the image capture when the camera is ready. If the camera hasn't initialized or is currently in the process of capturing another image, an exception will be thrown. To help mitigate exceptions, consider disabling the mechanisms that trigger photo capture until the Initialized event fires. To keep things simple in this example, we'll skip that step.

Figure 4 shows the code you need to add to the MainPage class.

Capturing a photo and saving it are asynchronous endeavors. When the CaptureImage method is called, a chain of events initiates and control is passed back to the UI. As shown in the event sequence diagram in **Figure 5**, there are two stages to each image capture. First, the camera sensor captures the photo, and then images are created based on the sensor data.

Saving a Photo After the sensor captures the photo, two image files are created in parallel, a full-size image file and a thumbnail. You're under no obligation to use both of them. Each is available as a JPG image stream from the e.ImageStream property in the arguments of the corresponding events.

The media library automatically creates its own thumbnails for display in the Pictures Hub of the device, so this example doesn't need the thumbnail version of the image. However, if you want to display a thumbnail in your own app, the e.ImageStream from the CaptureThumbnailAvailable event handler would be an efficient choice.

When the stream is available, you can use it to save the image to several locations. For example:

- Camera Roll folder: Use the MediaLibrary.SavePictureToCameraRoll method.
- Saved Pictures folder: Use the MediaLibary.SavePicture method.
- Isolated Storage: Use the IsolatedStorageFileStream.Write method.

In this example, we'll save the image to the camera roll folder. For an example of how to save an image to Isolated Storage, see the Basic Camera Sample in the Windows Phone SDK. Add the code in **Figure 6** to the MainPage class.

In the code in **Figure 6**, messages are sent to the UI before and after the image is saved to the Camera Roll folder. These messages are simply to help you understand what's going on; they're not required. The BeginInvoke method is needed to pass the message to the UI thread. If you didn't use BeginInvoke, a cross-threading exception would be thrown. For brevity, this method lacks error-handling code.

Handling Rotation When you save a picture to the media library, the correct orientation of the image will be noted in the file's EXIF information. The main concern of your app is how the preview from the camera is oriented in the UI. To keep the preview appearing in the

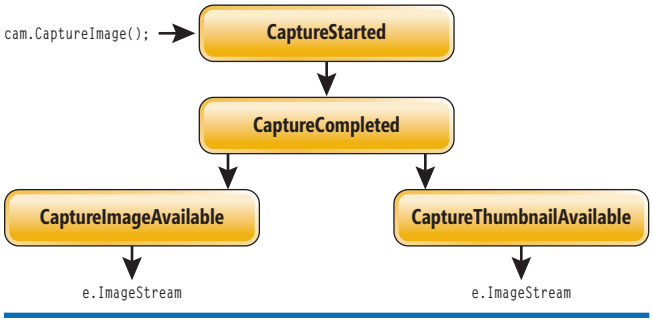


Figure 5 **The Image-Capture Event Sequence of the PhotoCamera Class**

5 YEARS OF EXCELLENCE

XCEED
DataGrid
for WPF

Mature, feature-packed, and lightning-fast. The most adopted and trusted WPF datagrid.

IBM®
U2 SystemBuilder™

"IBM U2 researched existing third-party solutions and identified Xceed DataGrid for WPF as the most suitable tool. The datagrid's performance and solid foundation take true advantage of WPF and provide the extensibility required for IBM U2 customers to take their applications to the next level in presentation design and styling."

Vincent Smith
U2 Tools Product Manager at IBM

Microsoft®
Visual Studio® Team System 2010

"Using Xceed DataGrid for WPF in Microsoft Visual Studio System 2010 helped us greatly reduce the time and resources necessary for developing all the data presentation feature we needed. Working with Xceed has been a pleasure."

Norman Guadagno
Director of Product Marketing
for Microsoft Visual Studio Team System

XCEED Professional Themes for WPF

Theme your entire app in minutes. Flawless styles for all official WPF controls.

XCEED DataGrid for Silverlight

Incredible streaming technology. Speed up your app and say goodbye to paging.

XCEED Ultimate ListBox for WPF

NEW

The world's first streaming listbox. Simple, drop-in upgrade to the WPF listbox.

XCEED Ultimate ListBox for Silverlight

NEW

Fast and fluid, with ground-breaking streaming technology.

Visual Studio PARTNER

XCEED MULTI-TALENTED COMPONENTS

Figure 6 **Saving an Image to the Camera Roll Folder**

```
void cam_CaptureImageAvailable(object sender,
  Microsoft.Devices.ContentReadyEventArgs e)
{
  photoCounter++;
  string fileName = photoCounter + ".jpg";

  Deployment.Current.Dispatcher.BeginInvoke(delegate()
  {
    txtMessage.Text = "Captured image available, saving picture.";
  });

  library.SavePictureToCameraRoll(fileName, e.ImageStream);

  Deployment.Current.Dispatcher.BeginInvoke(delegate()
  {
    txtMessage.Text = "Picture has been saved to camera roll.";
  });
}
```

correct orientation, rotate the viewfinder (the VideoBrush) as applicable. Rotation is achieved by overriding the OnOrientationChanged virtual method. Add the code in **Figure 7** to the MainPage class.

Without any adjustment to the viewfinder orientation, the viewfinder for a typical primary camera will appear oriented correctly only when the hardware shutter button is pointing up (LandscapeLeft). If you rotate the device such that the hardware shutter button is pointing down (LandscapeRight), the viewfinder must be rotated 180 degrees to display correctly in the UI. The PhotoCamera Orientation property is used here in case the physical orientation of the primary camera is atypical.

Declaring Application Capabilities Finally, when your application uses a camera, you must declare that it does so in the application manifest file, WMAppManifest.xml. No matter which camera is used, you'll need the ID_CAP_ISV_CAMERA capability. Optionally, you can use the ID_HW_FRONTCAMERA to designate that your app requires a front-facing camera:

```
<Capability Name="ID_CAP_ISV_CAMERA"/>
<Capability Name="ID_HW_FRONTCAMERA"/>
```

Your camera app won't run without the ID_CAP_ISV_CAMERA capability. If you haven't had a problem running it so far, it's because this capability is added to new Windows Phone projects automatically. If you're upgrading your app, though, you'll need to add it manually. ID_HW_FRONTCAMERA must always be added manually, but its lack won't prevent your app from running.

These capabilities help warn users who don't have a camera on their device, but nothing stops them from downloading and purchasing your app. For that reason, it's a good idea to make a trial version of your app available. Then, if users miss the warnings, they won't spend money only to learn that your app won't work as expected on their device. Your app ratings will thank you later.

If you haven't done so yet, press F5 and debug this simple camera app on your device. You can debug the app on the emulator, but you'll see only a black box moving around the screen because the emulator doesn't have a physical camera. When debugging with a device, keep in mind that you can't view your new images in the Picture Hub until you untether the device from your PC.

To go deeper, take a look at the Basic Camera Sample in the Windows Phone SDK. That sample demonstrates the full API for capturing photos: from adjusting flash and resolution settings to incorporating touch focus and the hardware shutter button.

Figure 7 **Overriding the OnOrientationChanged Virtual Method**

```
protected override void OnOrientationChanged
  (OrientationChangedEventArgs e)
{
  if (cam != null)
  {
    Dispatcher.BeginInvoke(() =>
    {
      double rotation = cam.Orientation;

      switch (this.Orientation)
      {
        case PageOrientation.LandscapeLeft:
          rotation = cam.Orientation - 90;
          break;
        case PageOrientation.LandscapeRight:
          rotation = cam.Orientation + 90;
          break;
      }

      viewfinderTransform.Rotation = rotation;
    });
  }
  base.OnOrientationChanged(e);
}
```

Accessing the Camera Preview Buffer

In the previous example, the frames from the camera preview buffer were streamed to the viewfinder. The PhotoCamera class also exposes the current frame of the preview buffer to allow pixel-by-pixel manipulation of each frame. Let's take a look at a sample from the Windows Phone SDK to see how we can manipulate frames from the preview buffer and display them on a writable bitmap in the UI.

The PhotoCamera class exposes the current frame of the preview buffer with the following "get preview" methods:

- GetPreviewBufferArgb32: Integer array of the current frame in ARGB format
- GetPreviewBufferYCbCr: Byte array of the current frame in YCbCr format
- GetPreviewBufferY: Byte array of the luminance plane only, in a similar format

ARGB is the format used to describe color in Silverlight for Windows Phone applications. YCbCr enables efficient image processing, but Silverlight can't use YCbCr. If you want to manipulate a YCbCr frame in your application, you have to convert the frame

Figure 8 **The Camera Grayscale Sample UI**

Less Pain, More Gain

check out infragistics.com/reporting

NetAdvantage® for Reporting

EXPORT TO EXCEL, WORD AND PDF

Export reports from the client and server side in the popular format of your choice!

DATA ACCESS SUPPORT

Create MVVM-friendly reports with data accessed from an SQL Server, Oracle or any Object Data Source.

DESIGN INTERFACE

Optimize your data presentation and build attractive reports with an integrated and easy-to-use design-time experience.

REPORT VIEWER

View pixel-perfect reports with vector graphics in our Silverlight, ASP.NET, WPF and Windows Forms report viewer.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales 800 231 8588 • Infragistics Europe Sales +44 (0) 800 298 9055 • Infragistics India +91 80 4151 8042 • @infragistics

Copyright 1996-2011 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc.

to ARGB before it can be displayed. For more information about these formats and color conversion, see the MSDN Library page, "Camera Color Conversion (YCbCr to ARGB) for Windows Phone," at wpdev.ms/colorconversion.

The Camera Grayscale Sample from the Windows Phone SDK (see **Figure 8**) demonstrates how to manipulate ARGB frames from the preview buffer and write them to a writable bitmap image in almost real time. In this sample, each frame is converted from color to grayscale. Note that the goal of this sample is to demonstrate ARGB manipulation; if your app needs only grayscale, consider using the GetPreviewBufferY method instead.

In the XAML file, an image tag is used to host the corresponding writable bitmap (the black-and-white image in the lower-left corner of the UI), like so:

```
<Image x:Name="MainImage"
       Width="320" Height="240"
       HorizontalAlignment="Left" VerticalAlignment="Bottom"
       Margin="16,0,0,16"
       Stretch="Uniform"/>
```

When a button is pressed to enable the grayscale conversion, a new thread is created to perform the processing; a writable bitmap, having the same dimensions of the preview buffer, is created and assigned as the source of the Image control:

```
wb = new WriteableBitmap(
        (int)cam.PreviewResolution.Width,
        (int)cam.PreviewResolution.Height);

this.MainImage.Source = wb;
```

The thread performs its work in the PumpARGBFrames method. There, an integer array named ARGBPx is used to hold a snapshot of the current preview buffer. Each integer in the array represents one pixel of the frame, in ARGB format. This array is also created with the same dimensions as the preview buffer:

```
int[] ARGBPx = new int[
    (int)cam.PreviewResolution.Width *
    (int)cam.PreviewResolution.Height];
```

While the "grayscale" feature of the sample is enabled, the thread copies the current frame in the preview buffer to the ARGBPx array. Here, phCam is the camera object:

```
phCam.GetPreviewBufferArgb32(ARGBPx);
```

Once the buffer has been copied to the array, the thread loops through each pixel and converts it to grayscale (see the sample for more details about how that's accomplished):

```
for (int i = 0; i < ARGBPx.Length; i++)
{
  ARGBPx[i] = ColorToGray(ARGBPx[i]);
}
```

Finally, before processing the next frame, the thread uses the BeginInvoke method to update the WriteableBitmap in the UI. The CopyTo method overwrites the WriteableBitmap pixels with the ARGBPx array, and the Invalidate method forces the WriteableBitmap to redraw, like so:

```
Deployment.Current.Dispatcher.BeginInvoke(delegate()
{
  // Copy to WriteableBitmap.
  ARGBPx.CopyTo(wb.Pixels, 0);
  wb.Invalidate();

  pauseFramesEvent.Set();
});
```

The WriteableBitmap class enables a wide range of creative possibilities. Now you can incorporate the camera preview buffer into your repertoire of visuals for the UI.

Recording Video

Although you can use the PhotoCamera class to stream the preview buffer to the UI, you can't use it to record video. For that, you'll need some classes from the System.Windows.Media namespace. In the final part of this article, we'll look at the Video Recorder Sample from the Windows Phone SDK (see **Figure 9**) to see how to record video to an MP4 file in Isolated Storage. You can find this sample on the SDK code samples page.

The primary classes for video recording are:

- CaptureDeviceConfiguration: Use to check availability of a video capture device
- CaptureSource: Use to start and stop video recording/preview
- VideoBrush: Use to fill Silverlight UI controls with a CaptureSource or PhotoCamera object
- FileSink: Use to record video to Isolated Storage when a CaptureSource object is running

In the XAML file, a Rectangle control is used to display the camera viewfinder:

```
<Rectangle
  x:Name="viewfinderRectangle"
  Width="640"
  Height="480"
  HorizontalAlignment="Left"
  Canvas.Left="80"/>
```

A Rectangle control isn't required to display video, however. You could use the Canvas control, as shown in the first example. The Rectangle control is used simply to show another way to display video.

At the page level, the following variables are declared:

```
// Viewfinder for capturing video.
private VideoBrush videoRecorderBrush;

// Source and device for capturing video.
private CaptureSource captureSource;
private VideoCaptureDevice videoCaptureDevice;

// File details for storing the recording.
private IsolatedStorageFileStream isoVideoFile;
private FileSink fileSink;
private string isoVideoFileName = "CameraMovie.mp4";
```

When a user navigates to the page, the InitializeVideoRecorder method starts the camera and sends the camera preview to the rectangle. After creating the captureSource and fileSink objects, the InitializeVideoRecorder method uses the static CaptureDeviceConfiguration object to find a video device. If no camera is available, videoCaptureDevice will be null:

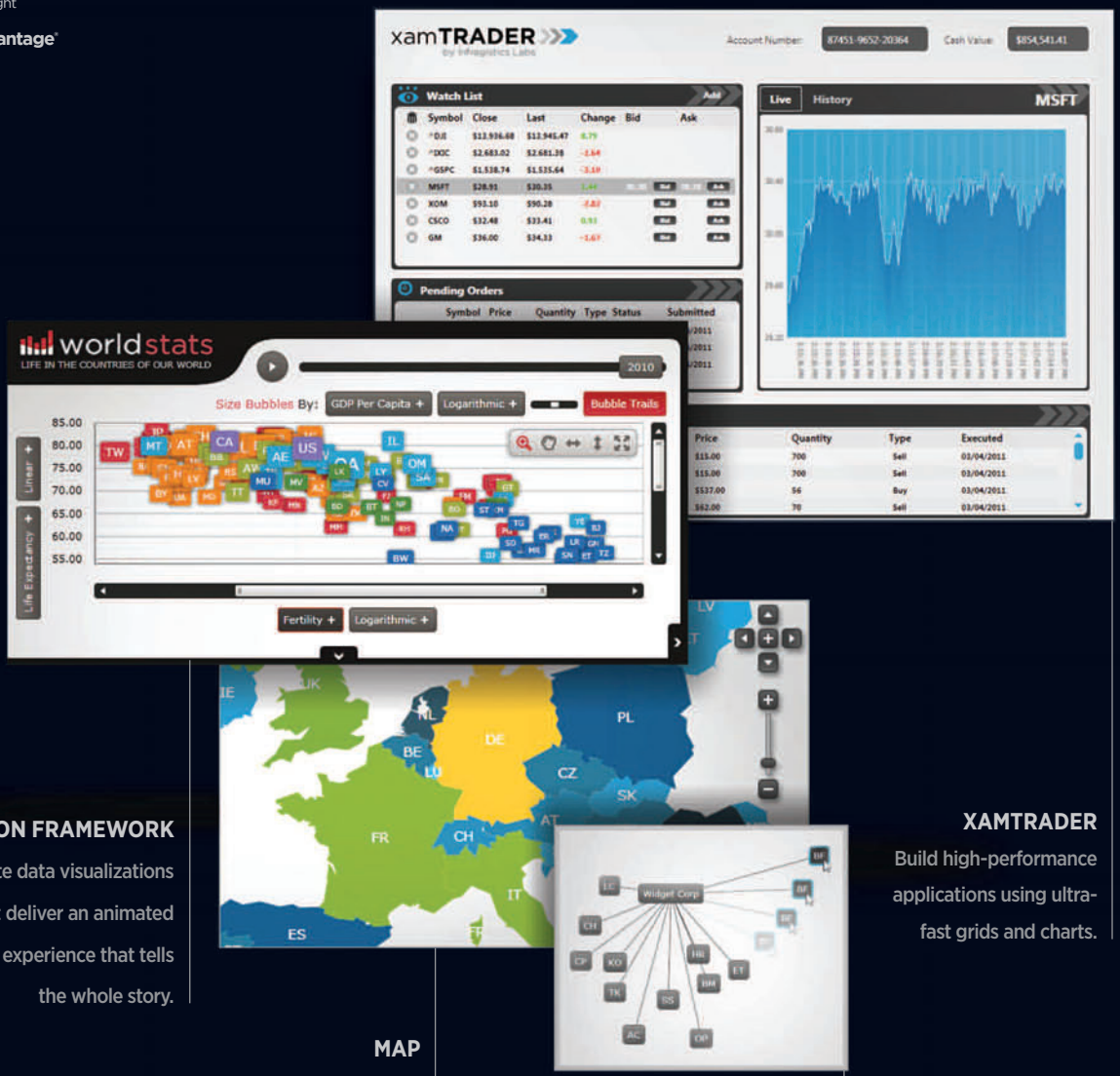
```
videoCaptureDevice = CaptureDeviceConfiguration.GetDefaultVideoCaptureDevice();
```

Figure 9 **The Video Recorder Sample UI**

XAML-IFY YOUR APPS

check out infragistics.com/xaml

- **NetAdvantage®** for Silverlight Data Visualization
- **NetAdvantage®** for WPF Data Visualization
- **NetAdvantage®** for Silverlight
- **NetAdvantage®** for WPF



MOTION FRAMEWORK

Create data visualizations that deliver an animated user experience that tells the whole story.

MAP

Ensure your geospatial data really goes places with a feature-laden, interactive Map Control for your applications.

XAMTRADER

Build high-performance applications using ultra-fast grids and charts.

NETWORK NODE

Help your users make the connection with visual representations of simple or complex network relationships.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales 800 231 8588 • Infragistics Europe Sales +44 (0) 800 298 9055 • Infragistics India +91 80 4151 8042 • @infragistics

Copyright 1996-2011 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc.

In Windows Phone 7.5, cameras are optional. Although they're common on today's devices, it's a best practice to check for them in your code. As **Figure 10** shows, videoCaptureDevice is used to check for the presence of a camera. If one is available, captureSource is set as the source of a VideoBrush named videoRecorderBrush, and videoRecorderBrush is used as the fill for the Rectangle control named viewfinderRectangle. When the Start method of captureSource is called, the camera begins sending video to the rectangle.

In this example, a helper method named UpdateUI manages button states and writes messages to the user. See the Video Recorder Sample for more details.

Although the fileSink object has been created, no video is being recorded at this point. This state of the application is referred to as video "preview." To record video, fileSink needs to be connected to captureSource *before* it's started. In other words, before you can record video, you need to stop captureSource.

When the user taps the record button in the video recorder sample, the StartVideoRecorder method starts the transition from preview to recording. The first step in the transition is stopping captureSource and reconfiguring the fileSink:

```
// Connect fileSink to captureSource.
if (captureSource.VideoCaptureDevice != null
    && captureSource.State == CaptureState.Started)
{
  captureSource.Stop();

  // Connect the input and output of fileSink.
  fileSink.CaptureSource = captureSource;
  fileSink.IsolatedStorageFileName = isoVideoFileName;
}
```

Although the CaptureSource and VideoBrush classes might sound familiar if you've developed applications for the Silverlight plug-in, the FileSink class is all new. Exclusive to Windows Phone applications, the FileSink class knows all about writing to Isolated Storage; all you need to do is provide the name of the file.

After fileSink has been reconfigured, the StartVideoRecorder method restarts captureSource and updates the UI:

```
captureSource.Start();

// Set the button states and the message.
UpdateUI(ButtonState.Ready, "Ready to record.");
```

When the user stops recording, to transition from recording to preview, captureSource needs to be stopped again before the fileSink is reconfigured, as shown in **Figure 11**.

The start-video-preview logic was isolated in another method to enable transition to preview from the video playback state (not covered in this article). Though I won't cover playback here, it's important to note that in Windows Phone, only one video stream can be running at a time.

The Video Recorder Sample features two separate video streams:

1. captureSource → videoRecorderBrush → viewfinderRectangle (Rectangle control)
2. isoVideoFile → VideoPlayer (MediaElement control)

Because only one stream can run at a time, this sample features a "dispose" method for each stream that can be called prior to the other stream running. In the DisposeVideoPlayer and DisposeVideoRecorder methods, the stream is stopped by calling the Stop method on the respective object (and setting the source of MediaElement to null). The CaptureSource and MediaElement objects don't actually implement the IDisposable interface.

Figure 10 **Displaying the Video Preview**

```
  // Initialize the camera if it exists on the device.
if (videoCaptureDevice != null)
{
  // Create the VideoBrush for the viewfinder.
  videoRecorderBrush = new VideoBrush();
  videoRecorderBrush.SetSource(captureSource);

  // Display the viewfinder image on the rectangle.
  viewfinderRectangle.Fill = videoRecorderBrush;

  // Start video capture and display it on the viewfinder.
  captureSource.Start();

  // Set the button state and the message.
  UpdateUI(ButtonState.Initialized, "Tap record to start recording...");
}
else
{
  // Disable buttons when the camera is not supported by the device.
  UpdateUI(ButtonState.CameraNotSupported, "A camera is not supported on
this device.");
}
```

Figure 11 **Transitioning from Recording to Preview**

```
// Stop recording.
if (captureSource.VideoCaptureDevice != null
&& captureSource.State == CaptureState.Started)
{
  captureSource.Stop();

  // Disconnect fileSink.
  fileSink.CaptureSource = null;
  fileSink.IsolatedStorageFileName = null;

  // Set the button states and the message.
  UpdateUI(ButtonState.NoChange, "Preparing viewfinder...");
  StartVideoPreview();
}
```

At this point, you might be thinking that the Camera Grayscale Sample seemed to have two videos going at the same time. In reality, there was only one video stream in that application: the stream from the PhotoCamera object to the VideoBrush control. The grayscale "video" was actually just a bitmap that was redrawn at a high rate of speed, based on individually manipulated frames from the camera preview buffer.

Wrapping Up

The camera API, new for Windows Phone 7.5, opens the door for a new breed of applications that solve problems and entertain in ways not possible with earlier versions of the OS. This article touched on only a few aspects of the API. For the complete reference, see the Camera and Photos section in the Windows Phone SDK documentation at wpdev.ms/cameraandphotos. ■

MATT STROSHANE *writes developer documentation for the Windows Phone team. His other contributions to MSDN Library feature products such as SQL Server, SQL Azure and Visual Studio. When he's not writing, you might find him out on the streets of Seattle, training for his next marathon. Follow him on Twitter at twitter.com/mattstroshane.*

THANKS to the following technical experts for reviewing this article:
Eric Bennett, Nikhil Deore, Adam Lydick and Jon Sheller

Deliver the Ultimate User Experience

check out infragistics.com/ultimate

NetAdvantage® ULTIMATE

FINANCIAL CHARTING

With support for multiple chart styles, and technical indicators built in, financial charting capabilities are on the money.

TREEMAP

Communicate the relative differences in data weight more effectively, with customizable color and flexible layouts.

OLAP AXIS CHART

Take your data to new depths with the seemingly endless drilldown capability of the OLAP Axis Chart.

OLAP GRID

Provide highly-interactive pivot grid functionality in all of your applications.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales 800 231 8588 • Infragistics Europe Sales +44 (0) 800 298 9055 • Infragistics India +91 80 4151 8042 • @infragistics

Copyright 1996-2011 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics, Inc.

Design Your Windows Phone Apps to Sell

Mark Hopkins

Good design is more than just adding pretty visuals to your application after you've implemented all the functionality. It's the process of deciding how users will interact with your app, as well as how it will look and function. User experience plays a huge part in determining how happy people will be with your app, so you shouldn't skimp on this step. Design should not be an afterthought.

I might be showing my age, but I remember almost every computer science class I took, starting with a lecture about the importance of planning your programs before beginning to code. We used tools such as flow charts drawn on actual paper with a real pencil (and usually a big eraser nearby). This was because computer time was expensive on that old hardware. You wanted to be sure you made the most of the time you had. Computer time is pretty cheap nowadays and there are amazing tools, such as Visual Studio and Expression Blend, that make it very easy to get something that looks pretty good up and running very quickly. Consequently, there's a tendency to sit down and just start coding. In this article, I'll talk about spending time up front designing your application so you can save time in the coding process and reap rewards in the Windows Phone Marketplace (windowsphone.com/marketplace).

I'll discuss what it means to intentionally design your Windows Phone app, which includes brainstorming, information architecture, prototyping, user research and iteration, all before you write a single line of code. I'll also note some of the tools available for these activities.

This article discusses:

- The importance of design
- Metro Design principles
- The design process
- Prototyping and user research

Technologies discussed:

Visual Studio, Expression Blend, Windows Phone SDK 7.1, SketchFlow

Why Design?

Look through the ratings and reviews on the Windows Phone Marketplace. One of the consistent complaints I find in reviews is when an app doesn't do what users expect. I've seen dismissive two-word reviews such as, "Doesn't work," or, often, much worse. Some of these complaints are valid and some are not. It might be that the reviewer doesn't understand how an application is intended to work. Is that the user's fault or the application's? Here's a review directly from the Marketplace:

"The UI is terrible and it runs really slow. It's unusable."

That review might be ambiguous, but I think it reflects the fact that people aren't willing to put a lot of effort into learning complicated applications on a phone. Nor are they willing to spend much time giving detailed feedback. Who can blame them? A phone is a casual-use device. Your app needs to be obvious and easy to use. With tens of thousands of apps available, users are unlikely to

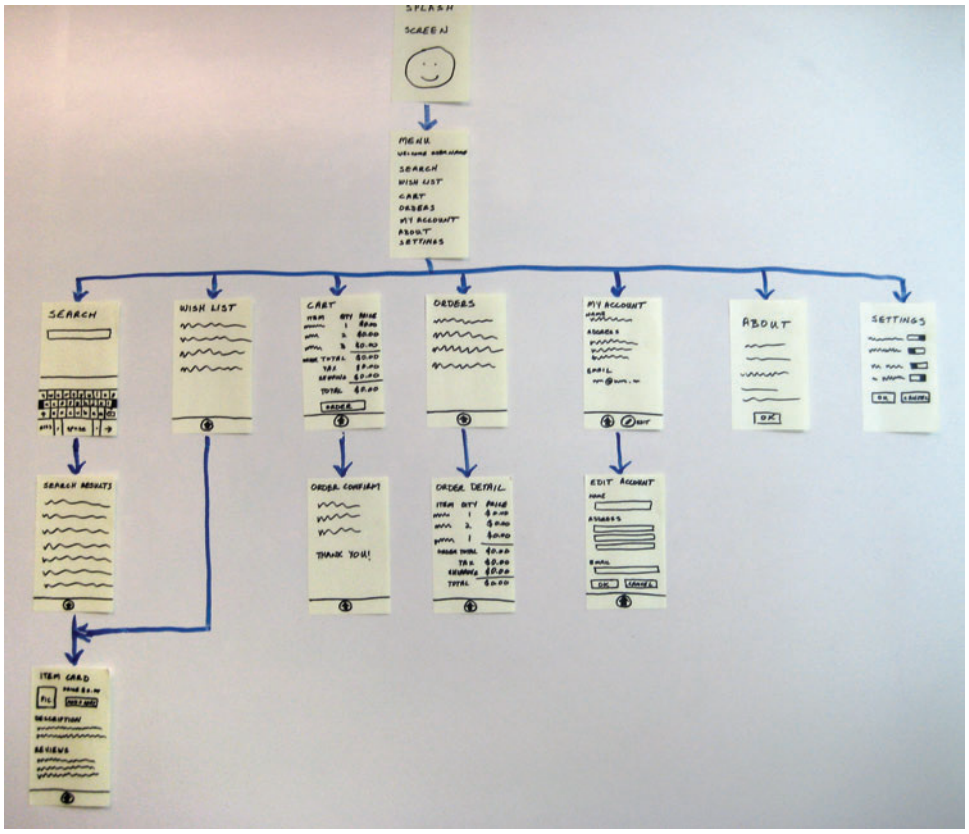


Figure 1 **Information Architecture Planning**

spend much time with one that doesn't give them the payoff they're seeking, whether that's solving a problem or being entertained while they wait in line at the bank.

Because there isn't much room for UI elements on a phone screen, you need to really think through how users will interact with your app. A well-designed, complete and easy-to-use application will generate sales. As people download and review your app in the Windows Phone Marketplace, those great reviews will fuel further downloads.

Metro Design Principles

Metro is the name of the design philosophy that goes into Windows Phone and into Windows 8. Much has been written about Metro, but I'd like to go over the Metro design principles briefly before I get into the actual process of designing an application.

- **Clean, Light, Open and Fast** Applications should be easy to understand at a glance. They should be highly responsive to user input. They should have a clean, open look with lots of white space.
- **Celebrate Typography** Words are common across all UI designs, and how they appear makes a difference. Type is beautiful. Not only is it pleasing to the eye, but it can also be functional. The right balance of weight and positioning can create a visual hierarchy. Moreover, well-placed type can direct users to more content.
- **Alive in Motion** Motion is life, and motion brings Windows Phone to life. Live Tiles, transitions and response to user input tie everything together. Transitions are an important part of user experience design. A good transition gives the user clues about context in your application.
- **Content, Not Chrome** Users are interested in content. Content should be elevated and everything else minimized. By removing as much chrome as possible, you bring the content into focus. This is particularly important on a small screen. The content is the UI and the user should be able to interact with it directly. The ability to resize an image using a pinch gesture is an example of this direct interaction.
- **Authentically Digital** Design explicitly for hand-held devices that use touch, a high-resolution screen, and simplified forms of interaction. In other words, be "authentically digital." Don't try to simulate analog controls such as knobs.

The Design Process

If you're lucky enough to work at a company that has a design department, get designers involved at the beginning. Your apps will reap the benefits of working with people who understand user experience design. Many of you probably work in small companies, though, or even develop phone apps as a side occupation on your own, so you'll have to handle the design yourself. Let's discuss the design process so you can include these practices as you create your Windows Phone masterpiece.

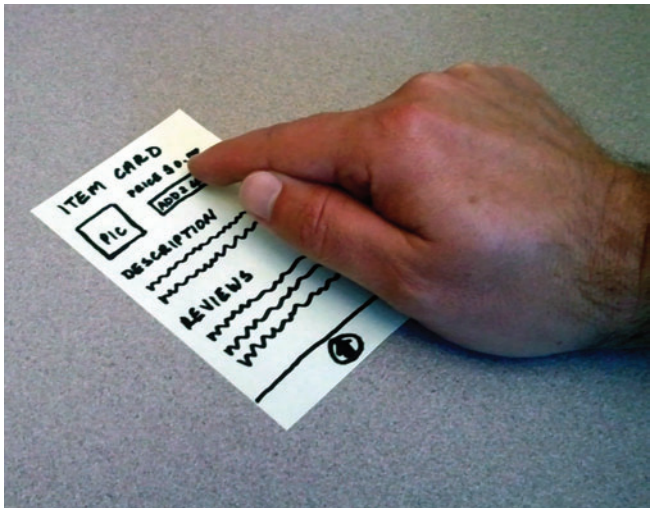


Figure 2 **Paper Prototype**

Brainstorm Be creative. You may already have an idea about an application you want to build or you might be trying to come up with one. In either case, brainstorming helps you explore ideas you might not have considered. And it's fun!

Try to brainstorm with others, if possible. If you're a lone developer creating apps by yourself, ask your family or a group of friends to do this with you. But be mindful of legal issues that could arise if you use someone else's idea. The point of brainstorming is to generate as many ideas as possible.

I'm going to assume you're familiar with the mechanics of brainstorming. But just to review, here are some guidelines for the process:

- Write down *everything*.
- No idea is too outrageous at this stage.
- Set a time limit to keep things moving.
- Don't deep dive on anything yet.

Your app could help people accomplish a task, or its purpose might be to entertain them. In any case, *you* are the storyteller. Consider these questions to help seed your brainstorming:

- What will your application do?
- Who is your application for?
- How does your application fit into the Marketplace?
- Where and when will your application be used?
- What kind of content will you display?
- How can your application leverage the hardware?
- How does your app idea compare with similar apps in the Windows Phone Marketplace and other smartphone application stores?

After you've come up with a great list of ideas, filter them through a list of constraints to help you narrow them down to something you'll actually create. Your questions might include:

- Do I have, or can I gain, the skills to realize the vision of this app?
- Can I develop this app in a reasonable time frame?
- Can I afford to develop this app?
- Can I leverage additional assets like online services?
- Can I partner with someone to make this app even better?
- Is there a lot of infrastructure needed for this app? For example, a streaming media service might be an awesome idea, but can I afford the server resources needed to get such a service off the ground?

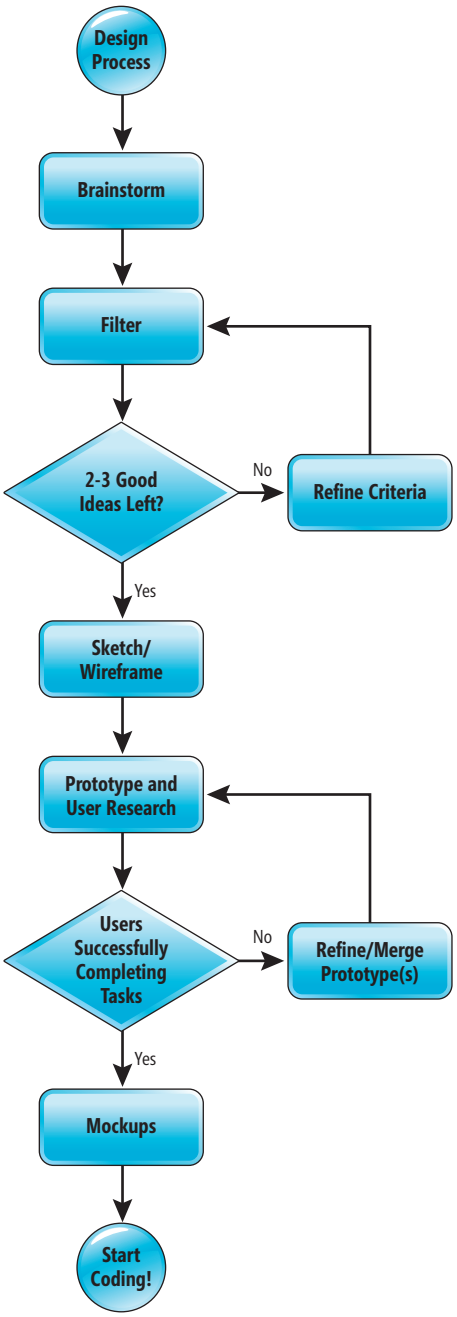


Figure 3 **The Design Process**

If you've done a good job on brainstorming, you should have a great list of ideas, so you'll probably have to be very critical to get your list down to two or three ideas you really want to move forward on.

Sketch and Wireframe Designing a prototype involves pulling together your brainstorming ideas. You might want to start by evaluating some similar apps that are already in the Marketplace. You'll probably discover both good and bad designs. Each offers good opportunities for learning.

Use a whiteboard or pencil and paper to move quickly. Lay out the navigation architecture for your app. Designers call this the *information architecture*, although that means something completely different to those of us in the content publishing world. No matter what you call it, this step can uncover inconsistencies in your design. It can also inspire ideas for making your app easier to use. I did a quick exercise with some sticky notes and my whiteboard (see **Figure 1**). This example is really simple, but it shows how this step might look. Each sticky note represents a page in an online shopping application.

The detail here isn't important; I drew up each of these pages in less than a minute. The point is to really think through the flow and navigation of your app. Sticky notes are great tools to quickly mock up the pages of an application. And because they're similar in size to an actual phone screen, they can impart the feel of an actual app.

Using a whiteboard to lay out the navigation makes it easy to move pages around and draw connections without investing too much time. This exercise quickly and obviously shows where there are holes in your planned interface. It also helps you estimate how much coding work you'll be undertaking to implement your vision.

Try to bring in other people and get their feedback on your information architecture, perhaps coworkers, or friends and family members if you're working on your own. Walk them through the design, but try not to overexplain. The purpose and navigation of your app should be obvious; keep refining until it is.

Prototyping and User Research Once you're happy with the pages and navigation of your design, it's time to create a prototype and start getting some feedback from users.

The prototype doesn't have to be a working application. Paper prototypes are excellent tools for quickly creating the look and feel of your app to present to users for usability testing (see **Figure 2**). Of course, if you're using paper, you'll need to help the user

understand how the application works. Do a YouTube search for "paper prototype" and you'll find many videos showing this process.

Once you have a couple of prototypes, invite users in to try them out. Ask users to state their goals within an application rather than to make specific suggestions about UI or interaction.

Try to answer the following questions about the functional elements of your app:

- Is it clear what your application does?
- Is it clear how to operate it?
- Are tasks intuitive in both purpose and operation?

This can be an incredibly powerful and rewarding process. I once watched a design team quickly create new designs on the fly, before the next group of usability test subjects even showed up. They could test out several designs in a single day. Talk about agile!

Iterate and Refine Continue to refine your prototype based on user feedback. Be careful not to get caught up in user requests for more and more features. While some user feedback might alert you to UI problems, most should be absorbed holistically. Keep your application focused and minimal.

Once you've refined your design to something you'd like to implement, move to the computer. Consider using a prototyping tool such as SketchFlow (microsoft.com/expression/products/Sketchflow_Overview) to start creating mockups of the actual page designs. Include enough detail so you won't need to make decisions during coding about:

- Visual elements: Are typography and content presented clearly, legibly and concisely? Is the display visually appealing? Theme Resources for Windows Phone (msdn.microsoft.com/library/ff769552(VS.92)) make it easier to adhere to Metro design principles, as well as user preferences, by providing predefined values for properties such as brushes, colors and fonts.
- Control elements: Are controls sized and spaced for easy touch operation? Note that in Visual Studio, the tools in the toolbox are already Metro-themed.
- Branding elements: Have you accurately reproduced colors and logos? Is all art compliant with copyrights?

Make sure you implement all the elements necessary to recreate the interactions you mapped out during prototyping. Confirm that the tasks and operations look and flow correctly based on your earlier usability testing.

The simple workflow chart in **Figure 3** visually represents the design process I've described.

Design Tools

It should now be obvious that you don't need to invest in a lot of expensive tools to do a good job of designing your app. Chances are you have some of these tools available right now.

- Paper and pencil: The original design tools, still powerful.
- Sticky notes: These are great "canvases" for phone page designs.
- Video camera: For recording usability tests and creating stop-motion animations of your designs to share with coworkers, friends, family.
- Windows Phone SDK 7.1: A free download (bit.ly/snlph6) that includes Visual Studio 2010 Express for Windows Phone, the Windows Phone Emulator and Expression Blend for Windows Phone.
- SketchFlow: Lets you quickly sketch up functional designs in Expression Blend.

Resources

There are many online resources that can help as you design your Windows Phone apps. The following documentation resources dive into the topics discussed here in much greater detail:

- User Experience Design Guidelines for Windows Phone (wpdev.ms/wpuxguide)
- Design Resources for Windows Phone (wpdev.ms/dsnrsrcs)
- Microsoft .toolbox (wpdev.ms/designtb)
- Alfred Astort's blog posts (wpdev.ms/alfreddesign)
- PhotoShop Design Templates for Windows Phone (wpdev.ms/dsntemplates)

You never know where inspiration might come from, so input is important. The following Twitter users often tweet about useful and interesting design and user research topics:

- @WPDesignTeam
- @corrinab
- @SusanToddUX
- @mkruzeniski
- @arturot
- @augustdlr

Wrapping Up

Once your design is finalized, it's time to start coding. Because all the design work has been done up front, you'll be free to concentrate on the logic needed to implement the functionality you've already verified through usability testing. This process will save you time because you won't have to re-architect your app in the middle of implementation. And because you'll have already tested your application with actual users, you'll be more likely to end up with good reviews in the Windows Phone Marketplace.

The Windows Phone SDK tries to set you up for success with themed controls and Visual Studio templates that include headers, theme resources and so on. You still need to do the intellectual work up front, but the tools will help you create a nice Metro app once you move to the implementation phase.

This article is just the tip of the iceberg when it comes to design. I hope I've piqued your interest and helped you consider how this process can raise the quality of your next Windows Phone app. A well-designed app creates a feedback loop in the Windows Phone Marketplace that leads to more downloads and more sales. Invest more thought in the design of your apps and may they be wildly successful. ■

Mark Hopkins *is a senior programming writer on the Windows Phone Developer Documentation Team. He has been employed at Microsoft since 1992 working on developer-focused products including developer support, Visual C++, MFC, Windows Platform SDK, Internet Explorer SDK, Tablet PC SDK, Surface SDK and Windows Phone SDK.*

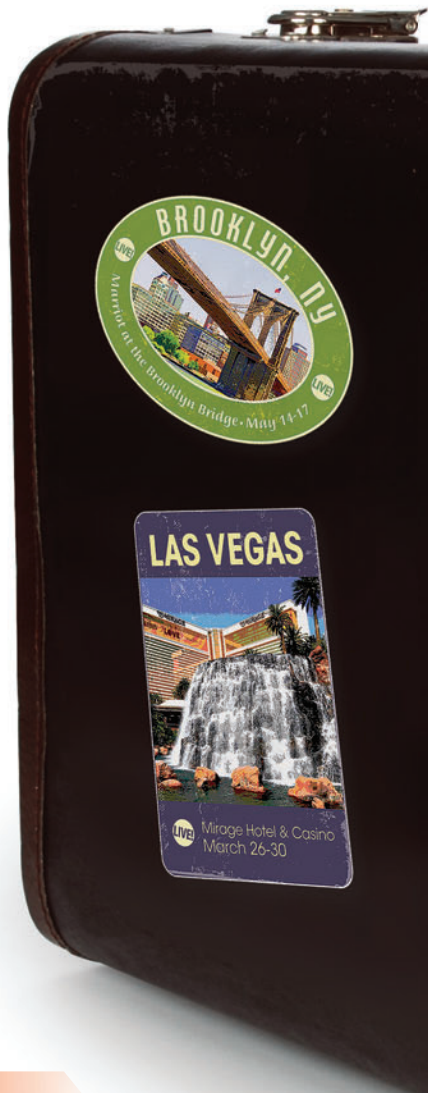
Thanks to the following technical experts for reviewing this article: *Robert Lyon, Cheryl Simmons and Matt Stroshane*

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS
2012

CODE ON THE ROAD

Visual Studio Live! is Coming to a City Near You

Visual Studio Live! features code-filled days, networking nights and unbiased training for developers, software architects and designers. Led by both independent industry experts and Microsoft insiders, these multi-track events include focused, cutting-edge education on the Microsoft Platform that you'll be ready to implement as soon as you get back to the office.

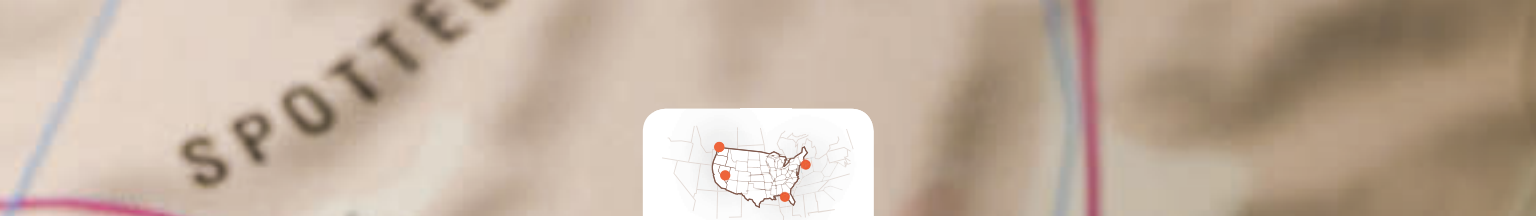


We now have FOUR awesome locations to choose from: **Las Vegas**, **New York**, **Microsoft Headquarters (Redmond, WA)** and **Orlando**. Pick your favorites and save the dates for 2012!

Learn more about Visual Studio Live! Events **vslive.com**

SUPPORTED BY:
Microsoft msdn Microsoft Visual Studio Visual Studio MAGAZINE

PRODUCED BY:
1105 MEDIA



YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

Visual Studio Live! Las Vegas

March 26-30, 2012

Mirage Resort and Casino, Las Vegas, NV

Registration now open!
vslive.com/lasvegas

Visual Studio Live! New York

May 14-17, 2012

New York Marriott at the Brooklyn Bridge

Registration now open!
vslive.com/newyork

Visual Studio Live! Redmond

August 6-10, 2012

Microsoft Headquarters in Redmond, WA

Registration now open!
vslive.com/redmond

Visual Studio Live! Orlando *TBD—Fall 2012*

Using HTML5 Canvas for Data Visualization

Brandon Satrom

In the early days online, when the Web was little more than a collection of static text and links, there was growing interest in supporting other types of content. In 1993, Marc Andreessen, creator of the Mosaic browser, which would evolve into Netscape Navigator, proposed the IMG tag as a standard for embedding images inline with the text on a page. Soon after, the IMG tag became the de facto standard for adding graphical resources to Web pages—a standard that's still in use today. You could even argue that, as we've moved from the Web of documents to the Web of applications, the IMG tag is more important than ever.

Media, in general, is certainly more important than ever, and though the need for media on the Web has evolved over the past 18 years, the image has remained static. Web authors have increasingly sought to use dynamic media such as audio, video, and interactive animations in their sites and applications and, until recently, the primary solution was a plug-in like Flash or Silverlight.

Now, with HTML5, media elements in the browser get a nice kick in the pants. You've probably heard of the new Audio and Video tags, both of which allow these types of content to function as first-class citizens in the browser, no plug-ins required. Next month's article will cover both of these elements and their APIs in depth. You've probably also heard of the canvas element, a drawing surface with a rich set of JavaScript APIs that give you the power to create and manipulate images and animations on the fly. What IMG did for static graphical content, canvas has the potential to do for dynamic and scriptable content.

As exciting as the canvas element is, though, it suffers from a bit of a perception problem. Because of its power, canvas is usually demonstrated via complex animations or games, and while these do convey what's possible, they can also lead you to believe that working with canvas is complicated and difficult, something that should be attempted only for complex cases, like animation or games.

In this month's article, I'd like to take a step back from the glitz and complexity of canvas and show you some simple, basic uses of it, all with the goal of positioning the canvas as a powerful option for data visualization in your Web applications. With that in mind, I'll focus on how you can get started with canvas, and how to draw simple lines, shapes and text. Then I'll talk about how you

This article discusses:

- The HTML5 canvas
- Drawing lines, shapes and text
- Working with colors and gradients
- Working with images
- Using a canvas polyfill

Technologies discussed:

HTML5, Internet Explorer, Modernizr

Code download available at:

code.msdn.microsoft.com/mag201201HTML5

can work with gradients in your shapes, as well as how to add external images to a canvas. Finally, and as I've done throughout this series, I'll wrap up with a brief discussion on polyfilling canvas support for older browsers.

Introducing the HTML5 Canvas

According to the W3C HTML5 specification (w3.org/TR/html5/the-canvas-element.html), the canvas element "provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics or other visual images on the fly." Canvas is actually defined across two W3C specifications. The first is as a part of the HTML5 core specification, where the element itself is defined in detail. This specification covers how to use the canvas element, how to obtain its drawing context, APIs for exporting canvas content and security considerations for browser vendors. The second is the HTML Canvas 2D Context (w3.org/TR/2dcontext), which I'll get to in a moment.

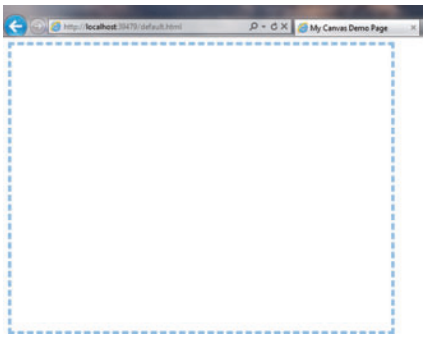


Figure 1 **A Blank, Styled Canvas Element**

Figure 2 **A Single Line on the Canvas**

Getting started with canvas is as simple as adding a <canvas> element to HTML5 markup, like so:

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>My Canvas Demo </title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <canvas id="chart" width="600" height="450"></canvas>
  </body>
</html>
```

Though I now have a canvas element in the DOM, placing this markup on the page does nothing, as the canvas element has no content until you add it. That's where the drawing context comes in. To show you where my blank canvas is located, I can use CSS to style it, so I'll add a dotted blue line around the blank element.

```
canvas {
    border-width: 5px;
    border-style: dashed;
    border-color: rgba(20, 126, 239, 0.50)
}
```

The result, when my page is opened in Internet Explorer 9+, Chrome, Firefox, Opera or Safari, is depicted in **Figure 1**.

When using canvas, you'll do most of your work in JavaScript, where the exposed APIs of a canvas drawing context can be leveraged to manipulate each pixel of the surface. To obtain the canvas drawing context, you need to get your canvas element from the DOM and then call the getContext method of that element.

```
var _canvas = document.getElementById('chart');
var _ctx = _canvas.getContext("2d");
```

GetContext returns an object with an API that you can use to draw on the canvas in question. The first argument to that method (in this case, "2d") specifies the drawing API that we want to use for the canvas. "2d" refers to the HTML Canvas 2D Context I mentioned earlier. As you might guess, 2D means that this is a two-dimensional drawing context. As of this writing, the 2D Context is the only widely supported drawing context, and it's what we'll use for this article. There's ongoing work and experimentation around a 3D drawing context, so canvas should provide even more power for our applications in the future.

Drawing Lines, Shapes, and Text

Now that we have a canvas element on our page and we've obtained its drawing context in JavaScript, we can begin to add content. Because I want to focus on data visualization, I'm going to use the canvas to draw a bar chart to represent the current month's sales data for a fictional sporting goods store. This exercise will require drawing lines for the axes; shapes and fills for the bars; and text for the labels on each axis and bar.

Let's start with the lines for the x- and y-axes. Drawing lines (or paths) with the canvas context is a two-step process. First, you "trace" the lines on the surface using a series of lineTo(x, y) and moveTo(x, y) calls. Each method takes x- and y-coordinates on the canvas object (starting from the top-left corner) to use when performing the operation (as opposed to coordinates on the screen itself). The moveTo method will move to the coordinates you specify, and lineTo will trace a line from the current coordinates to the coordinates you specify. For example, the following code will trace our y-axis on the surface:

```
// Draw y axis.
_ctx.moveTo(110, 5);
_ctx.lineTo(110, 375);
```

> When using canvas, you'll do most of your work in JavaScript.

If you add this code to your script and run it in the browser, you'll notice that nothing happens. Because this first step is merely a tracing step, nothing is drawn on the screen. Tracing merely instructs the browser to take note of a path operation that will be flushed to the screen at some point in the future. When I'm ready to draw paths to the screen, I optionally set the strokeStyle property of my context, and then call the stroke method, which will fill in the invisible lines. The result is depicted in **Figure 2**.

```
// Define Style and stroke lines.
_ctx.strokeStyle = "#000";
_ctx.stroke();
```

Because defining lines (lineTo, moveTo) and drawing lines (stroke) are decoupled, you can actually batch a series of lineTo and moveTo operations and then output those to the screen all at once. I'll do this for both the x- and y-axes and the operations that draw arrows as the end of each axis. The complete function for drawing the axes is shown in **Figure 3** and the result in **Figure 4**.

Figure 3 **The drawAxes Function**

```
function drawAxes(baseX, baseY, chartWidth) {
   var leftY, rightX;
   leftY = 5;
   rightX = baseX + chartWidth;

   // Draw y axis.
   _ctx.moveTo(baseX, leftY);
   _ctx.lineTo(baseX, baseY);

   // Draw arrow for y axis.
   _ctx.moveTo(baseX, leftY);
   _ctx.lineTo(baseX + 5, leftY + 5);
   _ctx.moveTo(baseX, leftY);
   _ctx.lineTo(baseX - 5, leftY + 5);

   // Draw x axis.
   _ctx.moveTo(baseX, baseY);
   _ctx.lineTo(rightX, baseY);

   // Draw arrow for x axis.
   _ctx.moveTo(rightX, baseY);
   _ctx.lineTo(rightX - 5, baseY + 5);
   _ctx.moveTo(rightX, baseY);
   _ctx.lineTo(rightX - 5, baseY - 5);

   // Define style and stroke lines.
   _ctx.strokeStyle = "#000";
   _ctx.stroke();
}
```

We have our axes, but we should probably label them to make them more useful. The 2D canvas context specifies APIs for adding text to canvas elements, so you don't need to fiddle with messy hacks like floating text over the canvas element. That said, canvas text doesn't provide a box model, nor does it accept CSS styles defined for page-wide text, and so forth. The API *does* provide a font attribute that works the same as a CSS font rule—as well as textAlign and textBaseline properties to give you some control over position relative to the provided coordinates—but other than that, drawing text on the canvas is a matter of picking an exact point on the canvas for the text you supply.

The x-axis represents products in our fictional sporting goods store, so we should label that axis accordingly:

```
var height, widthOffset;
height = _ctx.canvas.height;
widthOffset = _ctx.canvas.width/2;

_ctx.font = "bold 18px sans-serif";
_ctx.fillText("Product", widthOffset, height - 20);
```

In this code snippet, I'm setting the optional font property and providing a string to draw on the surface, along with the x- and y-coordinates to use as the start position of the string. In this example, I'll draw the word "Product" in the middle of my canvas, 20 pixels up from the bottom, which leaves room for the labels for each product on my bar chart. I'll do something similar for the y-axis label, which contains the sales data for each product. The result is depicted in **Figure 5**.

Now that we have a framework for our chart, we can add the bars. Let's create some dummy sales data for the bar chart, which I'll define as a JavaScript array of object literals.

```
var salesData = [{
   category: "Basketballs",
   sales: 150
}, {
   category: "Baseballs",
   sales: 125
}, {
   category: "Footballs",
   sales: 300
}];
```

With this data in hand, we can use fillRect and fillStyle to draw our bars on the chart.

fillRect(x, y, width, height) will draw a rectangle on the canvas at the x- and y-coordinates, with the width and height you specify. It's important to note that fillRect draws shapes starting from the top-left radiating outward, unless you specify negative width and height values, in which case the fill will radiate in the opposite direction. For drawing tasks like charting, that means we'll be drawing the bars from the top down, as opposed to the bottom up.

To draw the bars, we can loop through the array of sales data and call fillRect with the appropriate coordinates:

```
var i, length, category, sales;
var barWidth = 80;
var xPos = baseX + 30;
var baseY = 375;

for (i = 0, length = salesData.length; i < length; i++) {
   category = salesData[i].category;
   sales = salesData[i].sales;

   _ctx.fillRect(xPos, baseY - sales-1, barWidth, sales);
   xPos += 125;
}
```

In this code, the width of each bar is standard, while the height is taken from the sales property for each product in the array. The result of this code is seen in **Figure 6**.

Now, we have a chart that's technically accurate, but those solid black bars leave something to be desired. Let's spruce them up with some color, and then add a gradient effect.

Working with Colors and Gradients

When the fillRect method of a drawing context is called, the context will use the current fillStyle property to style the rectangle as its being

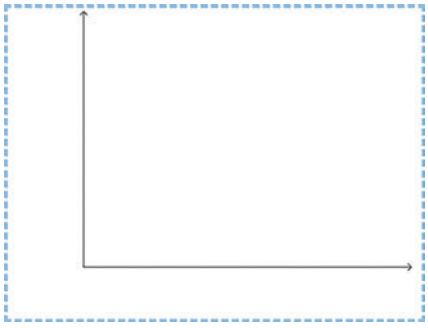


Figure 4 **Completed X- and Y-Axes**

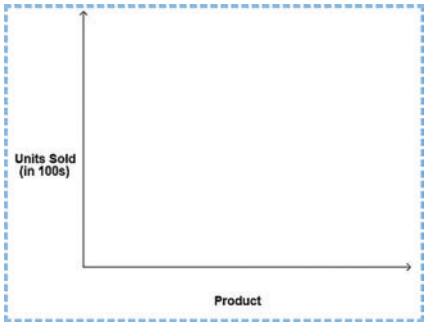


Figure 5 **Canvas with Text**

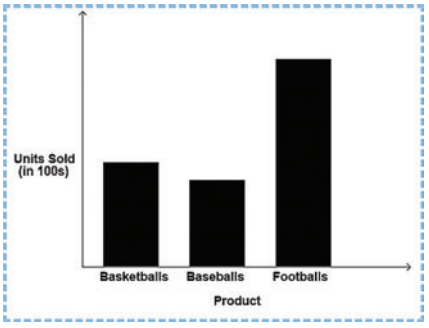


Figure 6 **Rectangles as Bar Chart Data**

drawn. The default style is a solid black, which is why our chart looks as it does in **Figure 6**. fillStyle accepts named, hexadecimal and RGB colors, so let's add some functionality to style each bar before it is drawn:

```
// Colors can be named hex or RGB.
colors = ["orange", "#0092bf", "rgba(240, 101, 41, 0.90)"];
...
_ctx.fillStyle = colors[i % length];
_ctx.fillRect(xPos, baseY - sales-1, barWidth, sales);
```

First, we create an array of colors. Then, as we loop through each product, we'll use one of these colors as the fill style for that element. The result is depicted in **Figure 7**.

This looks better, but fillStyle is very flexible and lets you use linear and radial gradients instead of just solid colors. The 2D drawing context specifies two gradient functions, createLinerGradient and createRadialGradient, both of which can enhance the style of your shapes through smooth color transitions.

For this example, I'm going to define a createGradient function that will accept the x- and y-coordinates for the gradient, a width and the primary color to use:

```
function createGradient(x, y, width, color) {
   var gradient;

   gradient = _ctx.createLinearGradient(x, y, x+width, y);
   gradient.addColorStop(0, color);
   gradient.addColorStop(1, "#efe3e3");

   return gradient;
}
```

After calling createLinearGradient with my start and end coordinates, I'll add two color stops to the gradient object returned by the drawing context. The addColorStop method will add color transitions along the gradient; it can be called any number of times with first parameter values between 0 and 1. Once I've set up my gradient, I'll return it from the function.

The gradient object can then be set as the fillStyle property on my context, in place of the hex and RGB strings I specified in the previous example. I'll use those same colors as my starting point, and then fade them into a light gray.

```
colors = ["orange", "#0092bf", "rgba(240, 101, 41, 0.90)"];
_ctx.fillStyle = createGradient(xPos, baseY - sales-1, barWidth,
colors[i % length]);
_ctx.fillRect(xPos, baseY - sales-1, barWidth, sales);
```

The result of the gradient option can be seen in **Figure 8**.

Working with Images

At this point, we have a pretty good-looking chart, which we've been able to render in the browser using a few dozen lines of JavaScript. I could stop here, but there's still one basic canvas API related to working with images I want to cover. Not only does canvas let you replace static images with script-based and interactive content, but you can also use static images to enhance your canvas visualizations.

For this demo, I'd like to use images as the bars on the bar chart. And not just any images, but pictures of the items themselves. With that in goal in mind, my Web site has a folder that contains JPG images for each product—in this case, basketballs.jpg, baseballs.jpg and footballs.jpg. All I need to do is position and size each image appropriately.

The 2D drawing context defines a drawImage method with three overloads, accepting three, five or nine parameters. The first parameter is always the DOM element image to draw. The simplest version of drawImage also accepts x- and y-coordinates on the canvas and draws the image as is in that location. You can also provide width and height values as the last two parameters, which will scale the image to that size prior to drawing it on the surface. Finally, the most complex use of drawImage allows you to crop an image down to a defined rectangle, scale it to a given set of dimensions and, finally, draw it on the canvas at the specified coordinates.

> The simplest course of action you can take for users whose browsers don't support canvas is to use a fallback element such as image or text.

Because the source images I have are large-scale images used elsewhere on my site, I'm going to take the latter approach. In this example, rather than calling fillRect for each item as I loop through the salesData array, I'll create an Image DOM element, set its source to one of my product images, and render a cropped version of that image onto my chart, as **Figure 9** shows.

Because I'm creating these images dynamically, as opposed to adding them manually to my markup at design time, I shouldn't assume that I can set the image source, then immediately draw that image to my canvas. To ensure that I draw each image only when it's fully loaded, I'll add my drawing logic to the onload event for the image, then wrap that code in a self-invoking function, which creates a closure with variables pointing to the correct product category, sales and positioning variables. You can see the result in **Figure 10**.

Using a Canvas Polyfill

As you may know, versions of Internet Explorer prior to 9, as well as older versions of other browsers, do not support the canvas element. You can see this for yourself by opening the demo project in Internet Explorer and hitting F12 to open the developer tools. From the F12 tools, you can change the Browser Mode to Internet

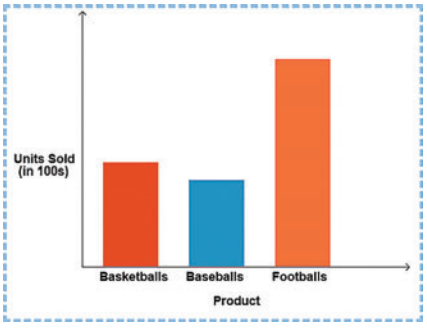


Figure 7 **Using fillStyle to Style Shapes**

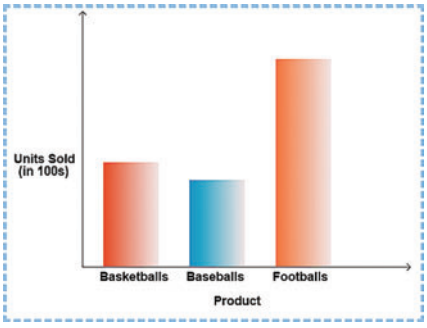


Figure 8 **Using Gradients in a Canvas**

Figure 9 **Drawing images on a Canvas**

```
// Set outside of my loop.
xPos = 110 + 30;
// Create an image DOM element.
img = new Image();
img.onload = (function(height, base, currentImage, currentCategory) {
  return function() {
    var yPos, barWidth, xPos;
    barWidth = 80;
      yPos = base - height - 1;

    _ctx.drawImage(currentImage, 30, 30, barWidth, height, xPos, yPos,
      barWidth, height);
      xPos += 125;
  }
})(salesData[i].sales, baseY, img, salesData[i].category);
img.src = "images/" + salesData[i].category + ".jpg";
```

Explorer 8 or Internet Explorer 7 and refresh the page. What you're likely to see is a JavaScript exception with the message "Object doesn't support property of method getContext." The 2D drawing context isn't available, nor is the canvas element itself. It's also important to know that, even in Internet Explorer 9, canvas isn't available unless you specify a DOCTYPE. As I mentioned in the first article of this series (msdn.microsoft.com/magazine/hh335062), it's always a good idea to use <!DOCTYPE html> at the top of all your HTML pages to ensure that the latest features in the browser are available.

The simplest course of action you can take for users whose browsers don't support canvas is to use a fallback element such as image or text. For instance, to display a fallback image to users, you can use markup that looks like this:

```
<canvas id="chart">
  <img id="chartIMG" src="images/fallback.png"/>
</canvas>
```

Any content you place inside of the <canvas> tag will be rendered only if the user's browser doesn't support canvas. That means that you can place images or text inside of your canvas as a simple, zero-checks fallback for your users.

If you want to take fallback support further, the good news is that a variety of polyfilling solutions exist for canvas, so you can feel comfortable using it with older browsers as long as you carefully vet potential solutions and stay aware of the limitations of a given polyfill. As I've stated in other articles in this series, your starting point for finding a polyfill for any HTML5 technology should be the HTML5 Cross Browser Polyfills page in the Modernizr wiki on GitHub (bit.ly/nZW85d). As of this writing, there are several canvas polyfills available, including two that fall back to Flash and Silverlight.

In the downloadable demo project for this article, I use explorercanvas (code.google.com/p/explorercanvas), which uses Internet Explorer-supported Vector Markup Language (VML) to create close approximations of canvas functionality, and canvas-text (code.google.com/p/canvas-text), which adds additional support for rendering text in older browsers.

As illustrated in previous articles, you can use Modernizr to feature-detect support for canvas (and canvastext) in a browser by calling Modernizr.canvas and then use Modernizr.load to asynchronously load explorercanvas when needed. For more information, see modernizr.com.

If you don't want to use Modenrizr, there's another way to conditionally add explorercanvas for older versions of IE: conditional comments:

```
<!--[if lt IE 9]>
  <script src="js/excanvas.js"></script>
  <script src="js/canvas.text.js"></script>
<![endif]-->
```

When Internet Explorer 8 or older versions encounter a comment formatted as such, they will execute the block as an if statement and include the explorercanvas and canvas-text script files. Other browsers, including Internet Explorer 10, will treat the entire block as a comment and ignore it altogether.

When evaluating a potential polyfill for your application, be sure to look into how much of the 2D drawing context a given polyfill supports. Few of them provide full support for every use, though nearly all can handle the basic cases we looked at in this article.

Though I couldn't cover everything here, there's a lot more you can do with canvas, from responding to click (and other) events and changing canvas data, to animating the drawing surface, rendering and manipulating images pixel-by-pixel, saving state, and exporting the entire surface as its own image. In fact, there are entire books on canvas out there. You don't have to be a game developer to experience the power of canvas, and I hope I convinced you of that as I walked through the basics in this article. I encourage you to read the specifications for yourself, and jump in to this exciting new graphics technology with both feet.

You can place images or text inside of your canvas as a simple, zero-checks fallback for your users.

If you're looking for more information on canvas support in Internet Explorer 9, check out the IE9 Developer Guide online (msdn.microsoft.com/ie/ff468705). Also, be sure to check out the Canvas Pad demos available at the IE Test Drive site (bit.ly/9v2zv5). For a list of a few other cross-browser polyfills for canvas, check out the complete polyfilling list at (hbit.ly/eBMoLW).

Finally, all of the demos for this article—which are available online—were built using WebMatrix, a free, lightweight Web development tool from Microsoft. You can try WebMatrix out for yourself at aka.ms/webm. ■

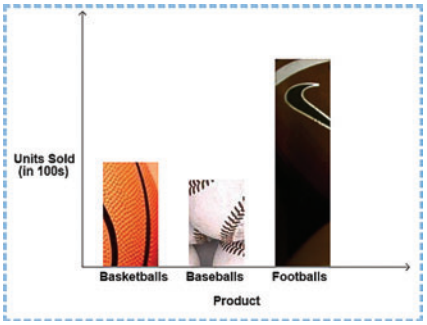


Figure 10 **Using Images on a Canvas**

BRANDON SATROM *works as a developer evangelist for Microsoft outside of Austin. He blogs at userinexperience.com and you can follow him on Twitter at twitter.com/BrandonSatrom.*

THANKS to the following technical experts for reviewing this article: *Jatinder Mann and Clark Sell*

ComponentSource®
The Definitive Source of Software Components
www.componentsource.com

NEW RELEASE

TX Text Control .NET for Windows Forms/WPF | from $1,045.59

TX TEXT CONTROL word processing components

Word processing components for Visual Studio .NET.

- Add professional word processing to your applications
- Royalty-free Windows Forms and WPF rich text box
- True WYSIWYG, nested tables, text frames, headers and footers, images, bullets, structured numbered lists, zoom, dialog boxes, section breaks, page columns
- Load, save and edit DOCX, DOC, PDF, PDF/A, RTF, HTML, TXT and XML

BEST SELLER

FusionCharts | from $195.02

InfoSoft Global
empowering human thoughts

Interactive Flash & JavaScript (HTML5) charts for web apps.

- Liven up your web applications using animated & data-driven charts
- Create AJAX-enabled charts with drill-down capabilities in minutes
- Export charts as images/PDF and data as CSV from charts itself
- Create gauges, dashboards, financial charts and over 550 maps
- Trusted by over 19,000 customers and 400,000 users in 110 countries

BEST SELLER

Spread for .NET Professional | from $1,439.04

GrapeCity PowerTools

Includes MultiRow, Stand-Alone Chart, Formula Provider and Runtime Spread Designer.

- Programmable embedded spreadsheet platform with royalty-free licensing
- Full import/export support for Excel documents
- Built-in support for Excel functions
- Flexible printing and PDF export options
- Built-in chart with hundreds of chart styles

BEST SELLER

Janus WinForms Controls Suite V4.0 | from $889.00

Janus systems

Add powerful Outlook style interfaces to your .NET applications.

- Includes Ribbon, Grid, Calendar view, Timeline and Shortcut bar
- Now features Office 2010 visual style for all controls
- Visual Studio 2010 and .NET Framework Client Profiles support
- Janus Ribbon adds Backstage Menus and Tab functionality as in Office 2010 applications
- Now features support for multiple cell selection

© 1996-2012 ComponentSource. All Rights Reserved. All prices correct at the time of press. Online prices may vary from those shown due to daily fluctuations & online discounts.

We accept purchase orders.
Contact us to apply for a credit account.

US Headquarters
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
USA

European Headquarters
ComponentSource
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
United Kingdom

Asia / Pacific Headquarters
ComponentSource
3F Kojimachi Square Bldg
3-3 Kojimachi Chiyoda-ku
Tokyo
Japan
102-0083

Sales Hotline - US & Canada:
(888) 850-9911
www.componentsource.com

GSA Schedule Contract GS-35F-0188R

Becoming a NuGet Author

Clark Sell

In the November issue, Phil Haack introduced NuGet, a new package management ecosystem for developers (msdn.microsoft.com/magazine/hh547106). NuGet is a project of the Outercurve Foundation, whose goal is to become a first-class package management system for the Microsoft .NET Framework. The project team consists mostly of Microsoft developers working in collaboration with the developer community. The introduction of NuGet to our development ecosystem gives .NET developers a way to consume, author and publish packages.

At first glance, NuGet might appear to be a tool just for the open source community, but that's only part of a larger story. NuGet was specifically designed not only to help distribute packages within the open source community, but also to deliver internal packages behind the firewall in an enterprise. This means you can use NuGet in a multifaceted way to install and update packages both from Microsoft and from the open source community at large, as well as your own internal servers.

In this article, I'll explore what it takes to become a NuGet package author. Incorporating NuGet into your development lifecycle isn't complicated, and it will yield substantial awesomeness. Once you've done this, your package consumption, creation and distribution problems will fade to a distant memory. Next month, I'll delve into what it takes to host your own packages.

This article discusses:

- The NuGet ecosystem
- The anatomy of a package
- The nuspec package manifest
- Publishing to NuGet.org
- Automating the build process with TFS NuGetter

Technologies discussed:

NuGet, Visual Studio, TFS NuGetter

Ecosystem Defined

Before starting on the journey to become a package author, let's briefly recap the larger ecosystem. NuGet, from the point of view of the package distributor, consists of a few essential components, mainly a command-line utility called NuGet.exe (nuget.codeplex.com/releases/view/58939) and a server to host the packages, such as the official NuGet Gallery, nuget.org. As last month's article demonstrated, you can interact with NuGet in three ways: using NuGet.exe, using NuGet inside of Visual Studio (the Package Manager Console window (View | Other Windows), and using the NuGet Package Explorer (npe.codeplex.com). Those utilities interact with one or more NuGet repositories. On the flip side, nuget.org is a public gallery you can use to store, host and publish NuGet packages. Nuget.org is built using another open source project known as NuGetGallery, which you can find at github.com/nuget/nugetgallery. I will talk more about how to host your own NuGet Gallery in the next issue.

As a package author, you can publish many different packages to a NuGet repository, and each package can have multiple versions. Nuget.org gives its customers a chance to read details about a package, install the package, contact the package owner and, in what should be rare cases, report abuse.

As the package author, you can control items such as version numbers, dependencies and how your package will be installed.

Getting Set Up

To publish a package, assuming you'll be using nuget.org as the repository, you need to sign up for an account on the NuGet Gallery. Becoming an author is easy: Just browse to the Contribute to NuGet Gallery section at nuget.org/contribute/index and select Get Started. Then, click on

Register now to get to the registration form and fill out the necessary information to create a new account. Nuget.org will send an e-mail with a URL where you can confirm your e-mail address and account.

After confirming your account, you can log on to the site and get your access key, a unique token that identifies you to the nuget.org repository and enables you to automate various package management tasks, such as pushing an update to your package.

For this article, I'll demonstrate both the NuGet.exe command line and the NuGet Package Explorer. One point to note: Once you download the command-line version, you'll probably want to update your system's path Environment variable to include its location. This makes it easy to use NuGet from anywhere on your system.

Anatomy of a Package

As noted in the last article, a NuGet package is an Open Packaging Convention (OPC) container file with the .nupkg file extension. The format for the package relies heavily on conventions, with a manifest file at the root known as the *nuspec* file. A sample directory structure initially might look like this:

```
Root Folder
|   package.manifest
+---lib
+---content
+---tools
```

As you can see, there are three folders at the root:

- **lib** contains all of the assemblies to be referenced
- **content** contains files and directories that are copied to the root of your target project
- **tools** is a place for custom Windows PowerShell scripts that might be run on installation of your package or every time the target project is loaded in Visual Studio

Of these, the lib folder is the most complex. It contains subfolders that correspond to framework dependencies, as shown here:

```
Root
| package.manifest
\---lib
    | MyFirstAssembly.dll
    \---net11
        | MySecondAssembly.dll
    \---net20
         | MySecondAssembly.dll
           \---sl
        | MySecondAssembly.dll
    \---netmf
         | MySecondAssembly.dll
+---content
+---tools
```

If your assembly works in all versions of the .NET Framework (rare indeed!), you need to include it only at the root of your lib folder, as with MyFirstAssembly.dll.

Given how rare that scenario is, the NuGet team strongly discourages the practice. It's better to make your assembly dependent on a specific framework version. To do so, add a folder for that framework version and include the correct version of the assembly in that folder. As you can see in the example folder, MySecondAssembly.dll has a different version for the .NET Framework 1.1, 2.0, Silverlight and the .NET MicroFramework. This ensures NuGet installs your package properly for the target frameworks.

When your customer installs your package, NuGet will install the correct assemblies based on the target framework for the project. In the previous example, let's assume your customer is trying to install your package into a project that targets version 4 of the .NET Framework. Because it's not listed as a framework in the sample lib folder, NuGet will take the closest framework version available and use that. In this example, it would take the assemblies found in the net20 folder and use those.

The content folder is a clone of the target project's root folder. Anything found in that folder will be copied as is into the target project. For example, if you wanted to copy some images into the target's /images folder, you'd need to include those images in the /content/images folder.

The tools folder includes any Windows PowerShell scripts that NuGet will invoke during package installation or when the project is opened, or that are used later by the customer. Once the folder is copied to the target project, it's added to the ` $env:Path (PATH) environment variable within the Visual Studio Package Manager Console.

NuGet has a built-in facility to automate the populating of your package—the files node. The files node is a way to explicitly list the files you want copied into your package structure when creating the .nupkg file. This helps automate the overall packing process. The file element is straightforward, defining src, target and exclude attributes. As you might guess, src defines the file you want copied; target defines the destination you want it copied to; and exclude defines what you don't want copied:

```
<files>
  <file src="bin\Debug\*.dll" target="lib" />
  <file src="bin\Debug\*.pdb" target="lib" />
  <file src="tools\**\*.*" exclude="*.log" />
</files>
```

If you have another process in place to create your package, you can ignore the files node in the .nuspec file.

The .nuspec File

The nuspec file is your package manifest. It's a simple XML file that defines your overall package and includes things like name, version number, package references and so on. To create your new manifest, NuGet.exe has a command called spec that you can use as a starting point:

```
> NuGet.exe spec
```

The spec command creates a new file called package.nuspec, a valid file that contains sample data. **Figure 1** shows an example file created by spec.

Figure 1 **Sample .nuspec File**

```
<?xml version="1.0"?>
  <package >
    <metadata>
      <id>Package</id>
      <version>1.0</version>
      <authors>csell5</authors>
      <owners>csell5</owners>
      <licenseUrl>http://LICENSE_URL_HERE_OR_DELETE_THIS_LINE</licenseUrl>
      <projectUrl>http://PROJECT_URL_HERE_OR_DELETE_THIS_LINE</projectUrl>
      <iconUrl>http://ICON_URL_HERE_OR_DELETE_THIS_LINE</iconUrl>
      <requireLicenseAcceptance>false</requireLicenseAcceptance>
      <description>Package description</description>
      <copyright>Copyright 2011</copyright>
      <tags>Tag1 Tag2</tags>
      <dependencies>
        <dependency id="SampleDependency" version="1.0" />
      </dependencies>
    </metadata>
  </package>
```

After this file is created, you replace the sample values. For most of the values, you only do this once, though some will change more often. For example, the id of your package shouldn't change after it's published, but the version number will change with every release.

You can also run nuget.exe spec against a Visual Studio project file (such as .csproj or .vbproj). In this case, the default values are already populated based on metadata in the project file. Here are some of the simpler elements of the .nuspec file:

- **id** unique identifier for the package
- **title** human-friendly title of the package
- **description** long description of the package
- **summary** short description of the package
- **licenseURl** a link to the license
- **copyrights** copyright details for the package

There are currently 28 different top-level elements. While the .nuspec file is pretty self-explanatory, you can find all of the details at bit.ly/lgQ4J4. Now let's take a look at a few of the more complex .nuspec elements.

Dependencies and References

We all know that managing dependencies can be difficult, especially when the chain of dependencies becomes long and intermingled. Let's say you've built PackageA. Your package happens to use PackageB, which can also be found in NuGet. Rather than including PackageB within your package, you just need to create a "dependency" on it. When someone starts to install your package, NuGet first inspects your nuspec file for its dependencies. It then looks at each dependency package and examines its dependencies and so on until it builds up a graph of every package it needs to download in order to fulfill all of the dependencies. It then downloads the entire graph of packages and installs them. This feature of NuGet drastically simplifies package creation and installation.

Let's look at some package dependencies defined as shown in the dependency node here:

```
<package>
<metadata>
<dependencies>
<dependency id="SampleDependency" version="1.0" />
  <dependency id="AnotherSampleDependency" version="[1.2,2.5)" />
</dependencies>
</metadata>
</package>
```

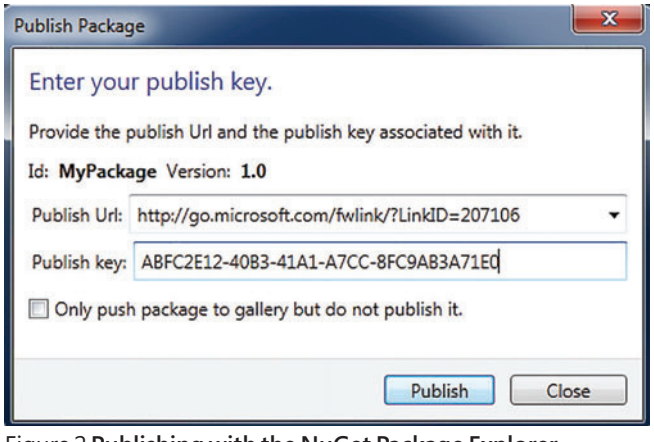


Figure 2 **Publishing with the NuGet Package Explorer**

You can list as many dependencies as you need. In each case, the id attribute indicates the package you have a dependency on, and the version attribute represents the version range you require. My example here shows a dependency on the SampleDependency project equal to version 1.0 or higher.

The NuGet version range specification gives you the ability to set the particular range of versions you allow. This looks something like version="[1.2, 2.5)", where the square bracket defines inclusion and the parenthesis defines exclusion. This example indicates that any package equal to or greater than 1.2 and less than 2.5 is allowed. NuGet will take the latest version found in that range. For detailed information about the version range specification, please visit bit.ly/qVXWxs.

In some cases, the person installing your package might need to program against types in a .NET Framework assembly. To add the appropriate reference, add the frameworkAssemblies node to the .nuspec file, detailing the list of required framework assemblies, like so:

```
<package>
  <metadata>
    <frameworkAssemblies>
      <frameworkAssembly assemblyName="System.Something"
targetFramework="net40" />
      <frameworkAssembly assemblyName="System.SomethingElse" />
    </frameworkAssemblies>
  </metadata>
</package>
```

Transformations

Many projects require more than just an assembly reference to work correctly. They may need a .config file change, or even some source code modified—and NuGet supports both of these scenarios natively. I'll focus on .config file transformations here. For more information about transformations, please see bit.ly/jqzry2.

During installation of your NuGet package, NuGet will run the transformation to add your new .config values. To make this happen, you need to add a transformation file to the content folder of your package. This is a valid XML file with the extension "transformation," whose filename matches the file to which you want to apply the transform. For example, to apply a transformation to a web.config file, you'd include a file named web.config.transformation.

Your transformation file should include only the sections of the configuration file you want added to target file. Let's say you want to add a new module to your customer's system.webServer section. Simply add that section in its entirety to the transformation file, like this:

```
<configuration>
  <system.webServer>
    <modules>
      <add name="NewModule" type="My.NewModule" />
    </modules>
  <system.webServer>
</configuration>
```

NuGet will not replace existing sections with the sections you add but rather merge them together. So if your target already had a modules section with its own module listed, the result of the two files being merged after installation would look something like this:

```
<configuration>
  <system.webServer>
    <modules>
      <add name="ExistingModule" type="Their.ExistingModule" />
      <add name="NewModule" type="My.NewModule" />
    </modules>
  <system.webServer>
</configuration>
```

As you can see, your module is added to the end of the existing stack of modules. If a user wanted to remove your package, just your changes would be removed (assuming you didn't make any changes to those sections), leaving the rest as it was in the first place.

Versioning

Versioning is at the core of anything we build. The NuGet package version refers to the package and not necessarily the assemblies contained inside (though it's customary to keep these in sync). You define the package version number in the .nuspec file, with a format of N.N.N.N, like so.

```
<package>
  <metadata>
    <version>1.2.3.4</version>
  </metadata>
</package>
```

There are a few properties in the .nuspec file where you can use a replacement token rather than just a static string. The version element is one of these. So instead of defining a static string like 1.2.3.4, you can insert a token, [$version$], which will later be replaced by NuGet.exe. With that token present, the version specified in the assembly's AssemblyVersionAttribute will be carried through to the .nuspec file:

```
<package>
  <metadata>
    <version>$version$</version>
  </metadata>
</package>
```

This is a great option if you want to keep the packages and version in sync, though there are plenty of reasons why you may not choose to do this.

Packing the Package

As mentioned earlier, a NuGet package is an OPC file with a .nupkg file extension. To create a package from the command line you simply call NuGet.exe with the pack command, passing it your .nuspec file:

```
> NuGet.exe Pack YourPackage.nuspec
```

As with spec, you can run pack against your project file as well. NuGet will build a complete NuGet package (.nupkg file) based solely on the metadata found within your .csproj or .vbproj file. If you've already created a .nuspec file, pack would use that .nuspec file:

```
> NuGet.exe pack [path]\MyProject.csproj
```

You've just created your first NuGet package, congratulations!

Symbol Support

Visual Studio has a great feature that lets developers step through source code on demand. NuGet supports this by giving package authors the ability to create and publish a symbol package. To create a symbol package, use the –symbols option when using pack:

```
> NuGet.exe pack MyProject.nuspec -symbols
> NuGet.exe pack MyProject.csproj -symbols
```

Pack will generate two .nupkg packages, MyProject.nupkg and MyProject.Symbols.nupkg. The .symbols.nupkg can later be pushed to SymbolSource.org using the command NuGet.exe push. For more information about creating symbol packages with NuGet, see bit.ly/jqzry2.

Publishing to NuGet.org

With your package created, it's now time to push it. Push is the NuGet command to publish your package to the server and it's used like most modern source control systems. Unlike the commands I've previously mentioned, push takes a number of arguments:

```
> NuGet.exe push <package path> [API key] [options]
```

- **Package path** The path to your package
  - Example c:\MyPackge\MyPackage.1.0.nupkg
- **API key** Your unique access token
  - Example: ABFC2E12-40B3-41A1-A7CC-8FC9AB3A71E0
  - Optional, could be set using the NuGet.exe setApiKey command
- **-source (src)** The server where the package is going.
  - Example: -source http://packages.nuget.org/v1/
  - Optional, unless you're pushing to an alternative place
- **-CreateOnly (co)** Pushes package to gallery but doesn't publish
  - Optional, default = false

The following sample command pushes the MyPackage package to NuGet:

```
> NuGet.exe push MyPackage.1.0.nupkg ABFC2E12-40B3-41A1-A7CC-8FC9AB3A71E0
```

You could also use NuGet Package Explorer, as shown in **Figure 2**.

If you built a symbol package, NuGet would automatically find it and push it to both repositories, nuget.org and symbolsource.org. If the target machine is set up to use symbolsource.org as a symbol source, the developer can now step into your package source files on demand from within Visual Studio.

You're published! If this is your second version of your package, that version will now become the default version. As explained in last month's article, when someone looks for package updates, your package will now be listed as one with an update.

Wrapping Up

Chances are your development team has some kind of build and deploy process in place. If you're like me, you're now starting to think about ways to integrate NuGet into that process. Clearly you could wrap all the commands I've shown here into your build process, but if you're already using Team Foundation Server (TFS), there's an easier way.

TFS NuGetter (nugetter.codeplex.com) is an open source project that extends the build process for TFS 2010, performing all of the necessary versioning, packaging and deployment functions in a customizable and repeatable way, with NuGet at its core. Regardless of your package's destination, TFS NuGetter will save you a great deal of time.

NuGet isn't a new concept to our industry, but for the .NET developer it may well seem groundbreaking. NuGet provides a much-needed package management facility, useful to everyone from garage developer to large enterprises. It gives you not only a place to publish your packages, but also a place for customers to discover your work. Publish your packages to NuGet and get found!

You can find all of the links used in this article and more at on.csell.net/BeANuGetAuthor. ■

CLARK SELL *works as a senior Web evangelist for Microsoft outside of Chicago. He blogs at csell.net, podcasts at DeveloperSmackdown.com and can be found on Twitter at twitter.com/csell5.*

THANKS to the following technical experts for reviewing this article: *David Ebbo, Phil Haack, Mark Nichols and Brandon Satrom*

Orchard Extensibility

Bertrand Le Roy

Most Web applications have a lot in common but at the same time exhibit a lot of differences. They all have static pages ("Terms of Use," "About Us" and so on). They present contents within a common layout. They have navigation menus. They might have search, comments, ratings and social network integration. But some are blogs, some sell books, some keep friends in touch and some contain hundreds of thousands of reference articles on your favorite technologies.

A Content Management System (CMS) aims at providing the common pieces while imposing no constraints on the type of site being built. It's a delicate exercise in extensibility.

This article discusses:

- Types and content items
- Building a part
- Database records
- The Part class
- Content part drivers
- Rendering shapes
- Packaging extensions
- Extensibility interfaces

Technologies discussed:

Orchard CMS

Code download available at:

bit.ly/u92283

The creators of the Orchard CMS (orchardproject.net)—that includes me—have opted for an approach that relies massively on composition and convention. In this article, I'll present a few examples of simple extensions to the system that should be a good starting point for your own modules.

A Dynamic Type System

No matter what CMS you use to build your site, there will be a central content entity that goes by different names. In Drupal, it's called a node, and in Orchard, it's a content item. Content items are atoms of content such as blog posts, pages, products, widgets or status updates. Some of them correspond to a URL, and some don't. Their schemas vary greatly, but what they have in common is they're the smallest content units on the site. Or are they?

Splitting the Atom

As developers, our first reaction is to identify content items as instances of classes (post, page, product or widget), which is correct to an extent. In the same way that classes are composed of members (fields, properties and methods), content types (the "classes" of content items) are themselves composite objects. Instead of being composed of simple properties with types that are themselves classes, they're composed of content parts, which are the atoms of behavior of your content. This is an important distinction, which I'll illustrate with an example.

A blog post typically is composed of a URL, title, date, rich text body, a list of tags and a list of user comments. None of those parts is specific to a blog post. What makes a blog post is the specific composition of the parts, not just any one of the parts.

Most blog posts have comments, but comments could also be used in a commerce site to implement reviews. Similarly, tags are potentially useful as a way to classify any content item, not just blog posts. The rich text body of a post is no different from the body of a page. The list goes on. It should be clear at this point that the unit of behavior on a Web site is smaller than the unit of content.

> No matter what CMS you use to build your site, there will be a central content entity that goes by different names.

Another fact about CMSes is that content types aren't fixed in advance. Blog posts used to be simple text, but they quickly became a lot more. They now routinely contain videos, podcasts or image galleries. If you blog about your travels around the world, maybe you'll want to add geolocation to your posts.

Again, content parts come to the rescue. You need a longitude and latitude? Extend the blog post type by adding a mapping part (we have several available in our module gallery: gallery.orchardproject.net). It quickly becomes obvious when you think about it that this operation of adding a part to an existing type will be performed most often by the owner of the site, not by a developer. Hence, it shouldn't be possible only by adding a complex property to a Microsoft .NET Framework type. It has to be metadata-driven and happen at run time so that we can build an admin UI to do it (see **Figure 1**).

This is the first way to extend Orchard: You can create and extend content types at will from the admin UI. Of course, anything that you can do from the admin UI, you can do from code, such as this:

```
item.Weld(part);
```

This code welds a part onto a content item dynamically. That's an interesting possibility, as it allows *instances* of types to be extended dynamically at run time. In dynamic languages, this is called a mix-in, but it's a concept that's almost unheard of in statically typed languages such as C#. This opens up new possibilities, but it isn't exactly the same thing as what we were doing from the admin UI. We also want to be able to add the part to the type once and for all instead of adding it to each instance, as shown here:

```
ContentDefinitionManager.AlterTypeDefinition(
  "BlogPost", ctb => ctb.WithPart("MapPart")
  );
```

This is actually exactly how the blog post content type is defined in the first place:

```
ContentDefinitionManager.AlterTypeDefinition("BlogPost",
  cfg => cfg
    .WithPart("BlogPostPart")
    .WithPart("CommonPart", p => p
      .WithSetting("CommonTypePartSettings.ShowCreatedUtcEditor", "true"))
      .WithPart("PublishLaterPart")
      .WithPart("RoutePart")
      .WithPart("BodyPart")
  );
```

You may notice in this code snippet that tags and comments seem to be missing from blog posts. This is one more example of careful separation of concerns. The blog module actually knows nothing of tags and comments, no more than the tags and comments modules do about blogs. A third party is responsible for putting them together.

Yummy Recipes

At setup, a recipe is executed that's responsible for this type of task. It's an XML description of the initial configuration of the site. Orchard comes with three default recipes: blog, default and core.

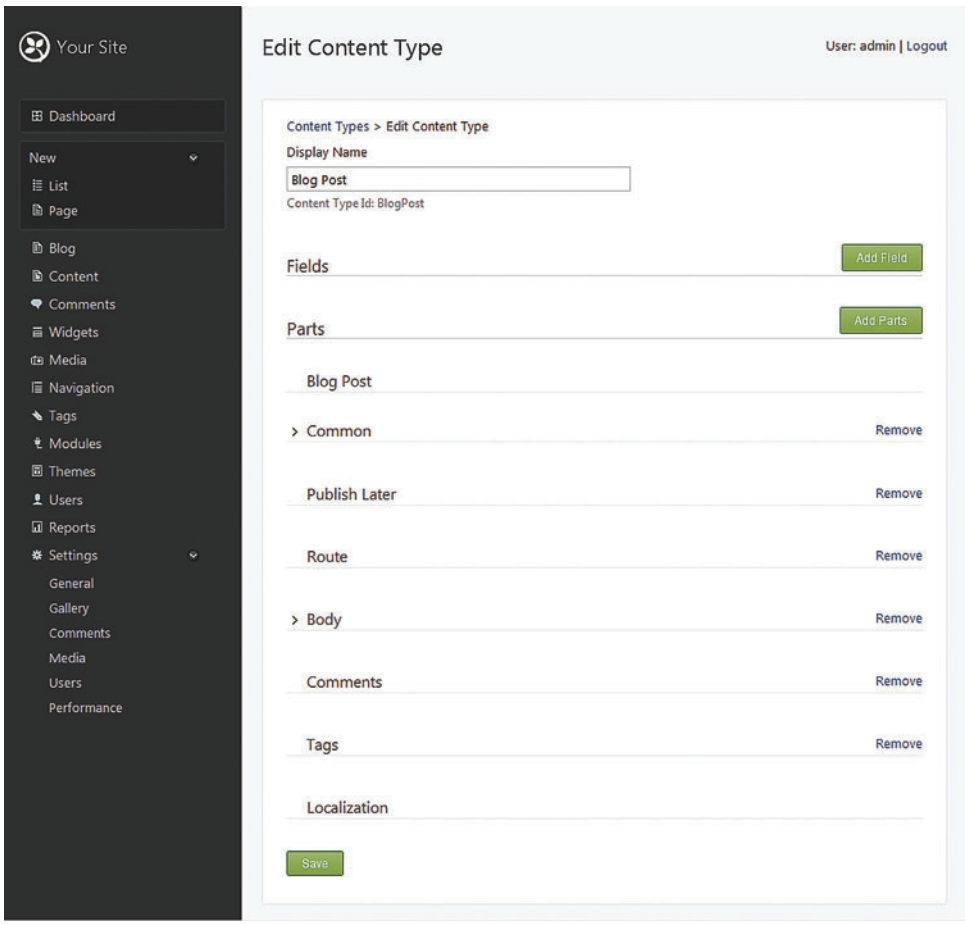


Figure 1 **The Orchard Content Type Editor**

The following code shows the part of the blog recipe that adds the tags and comments to blog posts:

```
<BlogPost ContentTypeSettings.Draftable="True" TypeIndexing.Included="true">
  <CommentsPart />
  <TagsPart />
  <LocalizationPart />
</BlogPost>
```

I've shown so far the various ways in which content parts can be composed into content items. My next step will be to explain how you can build your own parts.

Building a Part

To illustrate the process of building a new part, I'm going to rely on the example of the Meta feature from my Vandelay Industries module (download it from bit.ly/u92283). The Meta feature adds keywords and description properties for Search Engine Optimization (SEO) purposes (see **Figure 2**).

These properties will be rendered into the head section of the page as standard metatags that search engines understand:

```
<meta content="Orchard is an open source Web CMS built on ASP.NET MVC."
  name="description" />
<meta content="Orchard, CMS, Open source" name="keywords" />
```

The Record

The first piece of the puzzle will be a description of the way in which the data is going to be persisted into the database. Strictly speaking, not all parts need a record, because not all parts store their data in the Orchard database, but most do. A record is just a regular object:

```
public class MetaRecord : ContentPartRecord {
  public virtual string Keywords { get; set; }
  public virtual string Description { get; set; }
}
```

The MetaRecord class derives from ContentPartRecord. This isn't absolutely necessary, but it's definitely convenient as it gets some of the plumbing out of the way. The class has two string properties, for keywords and description. These properties must be marked virtual so the framework can "mix-in" its own logic to build the concrete class that will be used at run time.

The record's sole responsibility is database persistence, with the help of a declaration for the storage mechanism that can be found in the MetaHandler:

```
public class MetaHandler : ContentHandler
{
  public MetaHandler(
    IRepository<MetaRecord> repository) {
    Filters.Add(
      StorageFilter.For(repository));
  }
}
```

The storage also has to be initialized. Early versions of Orchard were inferring the database schema from the record classes, but guessing can only take you so far, and this has since been replaced with a more accurate migration system where schema modifications are explicitly defined, as shown in **Figure 3**.

The MetaRecord table is created with a name that the system will be able to map by convention to the MetaRecord class. It has the system columns for a content part record added by the call to the ContentPartRecord method, plus the Keywords and Description string columns that will automatically map by convention to the corresponding properties of the record class.

The second part of the migration method says that the new part will be attachable from the admin UI to any existing content type.

The Create method always represents the initial migration and

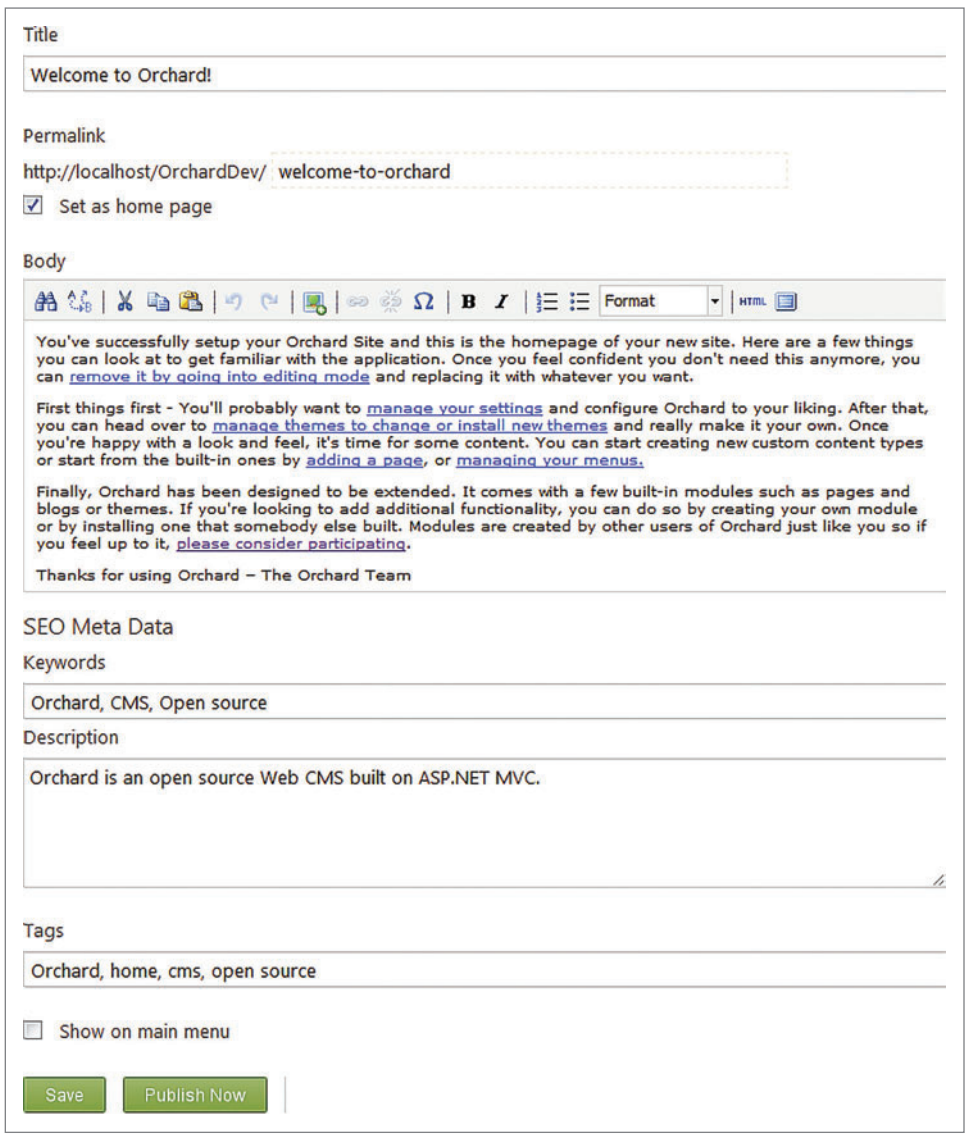


Figure 2 **The SEO Meta Data Editor**

Figure 3 **Explicitly Defined Schema Modifications**

```
public class MetaMigrations : DataMigrationImpl {

  public int Create() {
    SchemaBuilder.CreateTable("MetaRecord",
      table => table
        .ContentPartRecord()
        .Column("Keywords", DbType.String)
        .Column("Description", DbType.String)
    );

    ContentDefinitionManager.AlterPartDefinition(
      "MetaPart", cfg => cfg.Attachable());

    return 1;
  }
}
```

usually returns 1, which is the migration number. The convention is that in future versions of the module, the developer can add UpdateFromX methods, where X is replaced by the migration number from which the method operates. The method should return a new migration number that corresponds to the new migration number of the schema. This system allows for smooth, independent and flexible upgrades of all components in the system.

To represent the actual part, a separate class is used, which I'll look at now.

The Part Class

The representation of the part itself is another class that derives from ContentPart<TRecord>:

```
public class MetaPart : ContentPart<MetaRecord> {
  public string Keywords {
    get { return Record.Keywords; }
    set { Record.Keywords = value; }
  }

  public string Description {
    get { return Record.Description; }
    set { Record.Description = value; }
  }
}
```

The part acts as a proxy to the record's Keywords and Description properties as a convenience, but if it didn't, the record and its properties would still be available through the public Record property of the base ContentPart class.

Any code that has a reference to a content item that has the MetaPart part will be able to gain strongly typed access to the Keywords and Description properties by calling the As method, which is the analog in the Orchard type system of a CLR cast operation:

```
var metaKeywords = item.As<MetaPart>().Keywords;
```

The part class is also where you'd implement any specific behavior of the part's data. For example, a composite product could expose methods or properties to access its subproducts or compute a total price.

Behavior that pertains to user interaction (the orchestration code that would in a regular ASP.NET MVC application be found in the controller) is another matter. This is where drivers come in.

Figure 4 **The Driver's Display Method Prepares the Rendering of the Part**

```
protected override DriverResult Display(
  MetaPart part, string displayType, dynamic shapeHelper) {
  var resourceManager = _wca.GetContext().Resolve<IResourceManager>();
  if (!String.IsNullOrWhiteSpace(part.Description)) {
    resourceManager.SetMeta(new MetaEntry {
      Name = "description",
      Content = part.Description
    });
  }
  if (!String.IsNullOrWhiteSpace(part.Keywords)) {
    resourceManager.SetMeta(new MetaEntry {
      Name = "keywords",
      Content = part.Keywords
    });
  }
  return null;
}
```

The Driver

Each part in a content item has to get an opportunity to participate in the request lifecycle and effectively do the work of an ASP.NET MVC controller, but it needs to do it at the scale of the part instead of doing it at the scale of the full request. A content part driver plays the role of a scaled-down controller. It doesn't have the full richness of a controller in that there's no mapping to its methods from routes. Instead, it's made of methods handling well-defined events, such as Display or Editor. A driver is just a class that derives from ContentPartDriver.

> A content part driver plays the role of a scaled-down controller.

The Display method is what gets called when Orchard needs to render the part in read-only form (see **Figure 4**).

This driver is in fact a little atypical, because most drivers result in simple in-place rendering (more on that in a moment), whereas the Meta part needs to render its metatags in the head. The head section of an HTML document is a shared resource, so special precautions are necessary. Orchard provides APIs to access those shared resources, which is what you're seeing here: I'm going through the resource manager to set the metatags. The resource manager will take care of the rendering of the actual tags.

Figure 5 **The Editor Template for the MetaPart**

```
@using Vandelay.Industries.Models
@model MetaPart

<fieldset>
  <legend>SEO Meta Data</legend>

  <div class="editor-label">
    @Html.LabelFor(model => model.Keywords)
  </div>
  <div class="editor-field">
    @Html.TextBoxFor(model => model.Keywords, new { @class = "large text" })
    @Html.ValidationMessageFor(model => model.Keywords)
  </div>

  <div class="editor-label">
    @Html.LabelFor(model => model.Description)
  </div>
  <div class="editor-field">
    @Html.TextAreaFor(model => model.Description)
    @Html.ValidationMessageFor(model => model.Description)
  </div>

</fieldset>
```

Figure 6 **Injecting Dependencies through Constructor Parameters**

```
private readonly IRepository<ContentTagRecord> _contentTagRepository;
private readonly IContentManager _contentManager;
private readonly ICacheManager _cacheManager;
private readonly ISignals _signals;

public TagCloudService(
  IRepository<ContentTagRecord> contentTagRepository,
  IContentManager contentManager,
  ICacheManager cacheManager,
  ISignals signals)
  _contentTagRepository = contentTagRepository;
  _contentManager = contentManager;
  _cacheManager = cacheManager;
  _signals = signals;
}
```

The method returns null because there's nothing to render in-place in this specific scenario. Most driver methods will instead return a dynamic object called a shape, which is analogous to a view model in ASP.NET MVC. I'll come back to shapes in a moment when the time comes to render them into HTML, but for the moment, suffice it to say they're very flexible objects where you can stick everything that's going to be relevant to the template that will render it, without having to create a special class, as shown here:

```
protected override DriverResult Editor(MetaPart part, dynamic shapeHelper) {
  return ContentShape("Parts_Meta_Edit",
    () => shapeHelper.EditorTemplate(
      TemplateName: "Parts/Meta",
      Model: part,
      Prefix: Prefix));
}
```

The Editor method is responsible for preparing the rendering of the editor UI for the part. It typically returns a special kind of shape that's appropriate for building composite edition UIs.

The last thing on the driver is the method that will handle posts from the editor:

```
protected override DriverResult Editor(MetaPart part,
  IUpdateModel updater, dynamic shapeHelper) {
  updater.TryUpdateModel(part, Prefix, null, null);
  return Editor(part, shapeHelper);
}
```

> Site settings are effectively content items in Orchard, which makes a lot of sense once you understand how multitenancy is managed in Orchard.

The code in this method calls TryUpdateModel to automatically update the part with data that was posted back. Once it's done with this task, it calls in to the first Editor method in order to return the same editor shape it was producing.

Rendering Shapes

By calling in to all the drivers for all the parts, Orchard is able to build a tree of shapes—a large composite and dynamic view model for the whole request. Its next task is to figure out how to resolve each of the shapes into templates that will be able to render them. It does so by looking at the name of each shape (Parts_Meta_Edit in the case of the Editor method) and attempting to map that to files in well-defined places of the system, such as the current theme's and the module's Views folders. This is an important extensibility point because it enables you to override the default rendering of anything in the system by just dropping a file with the right name into your local theme.

In the Views\EditorTemplates\Parts folder of my module, I dropped a file called Meta.cshtml (see **Figure 5**).

Everything's Content

Before I move on to other extensibility topics, I'd like to mention that once you understand the content item type system, you understand the most important concept in Orchard. Many important entities of the system are defined as content items. For example, a user is a content item, which enables profile modules to add arbitrary properties to them. We also have widget content items, which can get rendered into zones that a theme defines. This is how the search forms, blog archives, tag clouds and other sidebar UI get created in Orchard. But the most surprising use of content items might be the site itself. Site settings are effectively content items in Orchard, which makes a lot of sense once you understand how multitenancy is managed in Orchard. If you want to add your own site settings, all you have to do is add a part to the Site content type, and you can build an admin edition UI for it following the exact same steps that I outlined earlier. A unified extensible content type system is an extremely rich concept.

> A theme is typically a bunch of images, stylesheets and template overrides, packaged into a directory under the Themes directory.

Packaging Extensions

I've shown how to build your own parts for Orchard, and I've alluded to the concepts of modules and themes without defining these terms. In a nutshell, they're the units of deployment in the system. Extensions are distributed as modules, and the visual look and feel is distributed as themes.

A theme is typically a bunch of images, stylesheets and template overrides, packaged into a directory under the Themes directory. It also has a theme.txt manifest file at its root to define metadata such as the author of the theme.

Similarly, a module is a directory under the Modules directory. It's also an ASP.NET MVC area, with a few twists. For example, it

needs an additional module.txt manifest file that declares some metadata for the module, such as its author, Web site, feature names, dependencies or version number.

Being only one area of a larger site, a module needs to play nice with a few shared resources. For example, the routes that it uses must be defined by a class implementing IRouteProvider. Orchard will build the full route table from what was contributed by all modules. Similarly, modules can contribute to building the admin menu by implementing the INavigationProvider interface.

Interestingly, the code for the modules isn't usually delivered as compiled binaries (although this is technically possible). This was a conscious decision that we made to encourage module hacking, where you start from a module that you download from the gallery and tweak it to address your specific needs. Being able to tweak the code for anything is one of the strengths of a PHP CMS such as Drupal or WordPress, and we wanted to provide that same kind of flexibility in Orchard. When you download a module, you download source code, and that code gets dynamically compiled. If you make a change in a source file, the change gets picked up and the module is recompiled.

Dependency Injection

So far, I've focused on one type of extensibility, the type system, because that represents the majority of Orchard modules, but there are many other extensibility points—far too many, in fact, for me to list here. I still want to add a few things about the general principles that are at work throughout the framework.

One key thing to get right in a highly modular system is loose coupling. In Orchard, almost everything above low-level plumbing is a module. Even modules are managed by a module! If you want these modules to work as independently from one another as possible—if you want one implementation of a feature to be swappable with another—you can't have hard dependencies.

A key way to reach this goal is to use dependency injection. When you need to use the services from another class, you don't just instantiate it, as that would establish a hard dependency on that class. Instead, you inject an interface that this class implements, as a constructor parameter (see **Figure 6**).

This way your dependency is on the interface, not the class, and the implementation can be swapped without having to change your code. The specific implementation of the interface that gets injected is no longer the decision of the consumer of the interface. Control is inverse here, and it's the framework that makes that decision.

Of course, this isn't limited to the interfaces that Orchard defines. Any module can provide its own extensibility points by just declaring an interface that derives from IDependency. It really is that simple.

Some Other Extensibility Points

I've only scratched the surface of extensibility here. There are many interfaces that can be used in Orchard to extend the system in creative ways. One could even say that Orchard is essentially nothing but an extensibility engine. All pieces in the system are swappable and extensible.

Before I conclude this article, I'll mention some of the most useful interfaces in the system that you might want to check out. I don't have nearly enough room here to go into these in depth, but I can give you pointers, and you can go into the code and follow usage of the interfaces. This is a really great way to learn, by the way.

- **IWorkContextAccessor** enables your code to access the work context for the current request. The work context, in turn, provides access to the HttpContext, current Layout, site configuration, user, theme and culture. It also provides facilities to get an implementation of an interface, from those places where you can't do dependency injection or where you need to delay it until after construction.
- **IContentManager** provides everything you need to query and manage content items.
- **IRepository<T>** gives access to lower-level data access methods, for those times when IContentManager isn't enough.

One could even say that Orchard is essentially nothing but an extensibility engine. All pieces in the system are swappable and extensible.

- **IShapeTableProvider** enables a boatload of on-the-fly shape manipulation scenarios. Basically, you hook up to events about shapes, and from them you can create alternate shapes to be used in certain situations, transform shapes, add members to them, move them around in the layout and so on.
- **IBackgroundTask, IScheduledTask and IScheduledTaskHandler** are the interfaces to use if you need delayed or repeated tasks to be executed in the background.
- **IPermissionsProvider** enables your modules to expose their own permissions.

Learn More

Orchard is an extensibility powerhouse, and presenting all it has to offer in this limited space is a challenge. My hope is that I gave you the will to learn more about it and dive deep into it. We have a friendly and lively community on orchard.codeplex.com/discussions that will be happy to guide you and answer your questions. With so much to implement, and so many ideas to explore, here's a great chance to provide a significant contribution. ■

BERTRAND LE ROY *started his professional developer career in 1982 when he published his first video game. He released in 2002 what was probably the first CMS to run on ASP.NET. A year later, he was hired by the Microsoft ASP.NET team and moved to the United States. He has worked on ASP.NET versions 2.0 to 4 and ASP.NET AJAX; contributed to making jQuery an official part of the .NET developer's tool chest; and represents Microsoft in the OpenAjax Alliance steering committee.*

THANKS to the following technical expert for reviewing this article: *Sebastien Ros*

Securing Your ASP.NET Applications

Adam Tuliper

In the previous issue, I discussed the importance of building security into your Web applications and looked at some types of attacks, including SQL injection and parameter tampering, and how to prevent them (msdn.microsoft.com/magazine/hh580736). In this article, I'll delve into two more common attacks to help round out our arsenal of application protections—cross-site scripting (XSS) and cross-site request forgery (CSRF).

You might be wondering: Why not just use a production security scanner? Scanners are great tools for finding low-hanging fruit, and they're especially good at finding application and system configuration issues, but they can't know your application the way you do. Therefore, it's imperative that you become familiar with potential security issues and take the time to check your applications and build security into your software development lifecycle.

Cross-Site Scripting

What Is It? XSS is an attack in which script is maliciously injected into a user's browsing session, generally without the user's knowledge. These types of attacks have become quite familiar to many people because of incidents in which a large social networking site was struck and its users had messages posted they didn't authorize. If an attacker posts malicious script that he can cause the browser to execute, this script is executed in the context of the victim's session, essentially enabling the attacker to do anything he wants to the DOM—including showing fake login dialogs or stealing cookies. Such an attack could even install an HTML key logger on the current page to continually send input from that window to a remote site.

> XSS is exploited via several methods that all rely on having unescaped or improperly escaped output.

How Is It Exploited? XSS is exploited via several methods, all of which rely on having unescaped or improperly escaped output. Let's take the case of an application that needs to display a simple status message to the end user. Typically this message is passed on the query string as shown in **Figure 1**.

This technique is commonly used after a redirect to show the user some sort of status, such as the Profile Saved message in **Figure 1**. The message is read from the query string and written right

This article discusses a beta version of the Microsoft Anti-Cross Site Scripting Library (AntiXSS).

This article discusses:

- Cross-site scripting
- Cross-site request forgery

Technologies discussed:

ASP.NET, SQL Server, Microsoft Anti-Cross Site Scripting Library

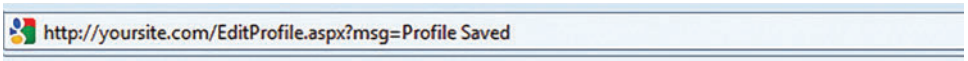


Figure 1 **Query String Message**

Figure 2 **Injecting Script into the URI**

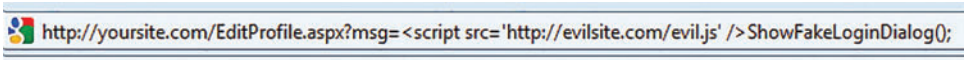


Figure 3 **Creating a Malicious Attack**

out to the page. If the output is not HTML-encoded, anyone can easily inject JavaScript in place of the status message. This type of attack is considered a *reflected* XSS attack, because whatever is on the query string gets rendered right back to the page. In a *persistent* attack the malicious script is stored, usually in a database or cookie.

> The key to the exploit here is that the results are not HTML-encoded.

In **Figure 1** you can see this URI takes a msg parameter. The Web page for this URI would contain code such as the following to simply write the variable to the page with no encoding:

```
<div class="messages"> <%=Request.QueryString["msg"]%></div>
```

If you replace "Profile Saved" with the script shown in **Figure 2**, the alert function will pop up in the browser from the script included on the query string. The key to the exploit here is that the results are not HTML-encoded and therefore the <script> tag is actually parsed as valid JavaScript by the browser and executed. This is clearly not what the developer here had in mind.

So an alert pops up—what's the big deal? Let's take this example a bit further, as shown in **Figure 3**. Note that I've shortened the attack here to make the point; this syntax isn't exactly correct but a slight modification would turn this into a real attack.

An attack like this would cause a fake login dialog to be shown to users, where upon they would blithely enter their credentials. In this case, a remote script is downloaded, which is generally allowed by the default browser security settings. This type of attack can in theory happen anywhere a string that hasn't been encoded or sanitized can be echoed back to a user.

Figure 4 shows an attack using similar script on a development site that allows users to leave comments about its products. Instead of leaving a real product review, however, someone entered malicious JavaScript as a comment. This script now displays a login dialog to every single user reaching the Web page and collects and sends the credentials to a remote site. This is a persistent XSS attack; the script is stored in the database and repeated for everyone who visits the page.

Another way XSS is exploited is by using HTML elements, as when dynamic text is allowed in HTML tags, like so:

```
<img onmouseover=alert([user supplied text])>
```

If an attacker injects text such as "onmouseout=alert(document.cookie)," this creates the following tag in the browser that accesses the cookie:

```
<img onmouseover=alert(1) onmouseout=alert(document.cookie) >
```

There's no "<script>" tag to potentially filter input on and nothing to escape, but this is a completely valid piece of JavaScript that can read a cookie—potentially an authentication cookie. There are case-specific ways to make this safer, but because of the risk, it's best not to allow any user input from reaching this inline code here.

How Do You Prevent XSS? Following these rules strictly will help prevent most if not all XSS attacks in your application:

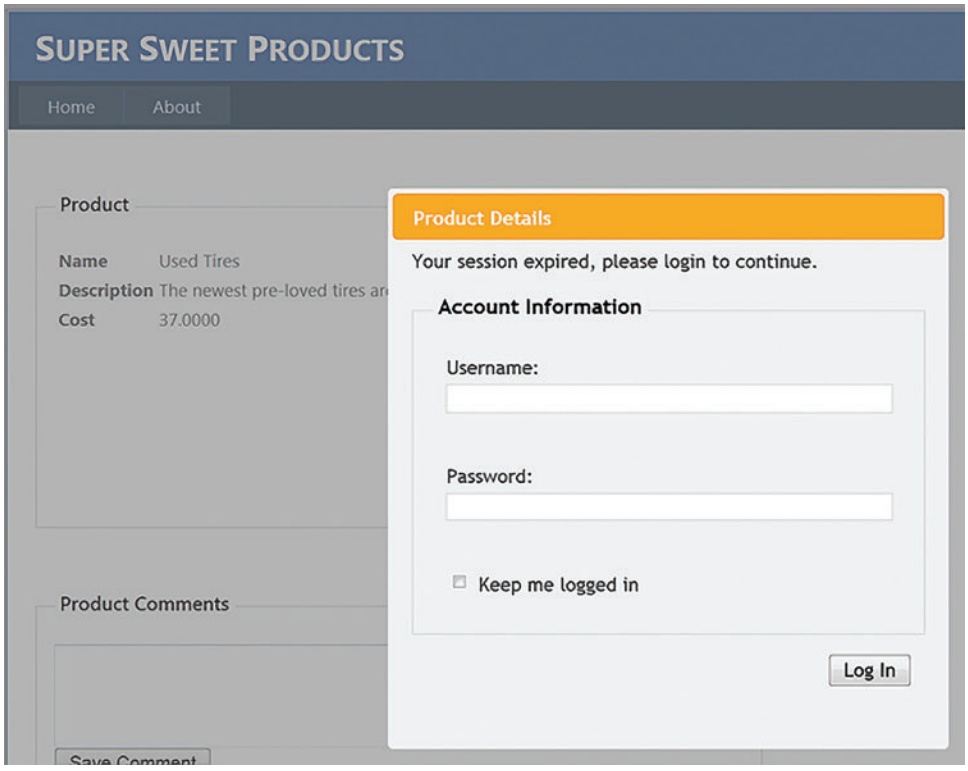


Figure 4 **A Persistent XSS Attack Showing a Fake Dialog**

Figure 5 **Server Error from Unencoded HTML**

1. Ensure all of your output is HTML-encoded.
2. Don't allow user-supplied text to end up in any HTML element attribute string.
3. Prevent the use of Internet Explorer 6 by your application by checking Request.Browser as outlined at msdn.microsoft.com/library/3yekbd5b.
4. Know your control's behavior and whether it HTML encodes its output. If it doesn't, encode the data going to the control.
5. Use the Microsoft Anti-Cross Site Scripting Library (AntiXSS) and set it as your default HTML encoder.
6. Use the AntiXSS Sanitizer object (this library is a separate download and is addressed later in this article) to call GetSafeHtml or GetSafeHtmlFragment before saving HTML data to the database; don't encode the data before saving.
7. For Web Forms, don't set EnableRequestValidation=false in your Web pages. Unfortunately, most user group postings on the Web advise disabling this setting if there's an error. The setting is there for a reason and will stop the request if the character combination "<X," for example, is posted back to the server. If your controls are posting HTML back to the server and receiving the error shown in **Figure 5**, you should ideally encode the data before you post it to the server. This is a common scenario with WYSIWYG controls, and most modern versions will properly encode their HTML data before posting back to the server.
8. For ASP.NET MVC 3 applications, when you need to post HTML back to your model, don't use ValidateInput(false) to turn off Request Validation. Simply add [AllowHtml] to your model property, like so:

```
public class BlogEntry
{
  public int UserId {get;set;}
  [AllowHtml]
  public string BlogText {get;set;}
}
```

Some products try to detect <script> and other word combinations or regular expression patterns in a string to try to detect XSS. These products can provide additional checks but aren't completely reliable because of the many variants attackers have created. Take a look at the XSS Cheat Sheet at ha.ckers.org/xss.html to see how difficult detection can be.

To understand the fixes, let's suppose an attacker has injected some script that ended up in a variable in our application from either the query string or a form field as shown here:

```
string message = Request.QueryString["msg"];
```

or:

```
string message = txtMessage.Text;
```

Note that even though a TextBox control HTML encodes its output, it doesn't encode its Text property when you read it in from code. With either of these lines of code, you're left with the following string in the message variable:

```
message = "<script>alert('bip')</script>"
```

In a Web page containing code similar to the following, the JavaScript will be executed in the user's browser simply because this text was written to the page:

```
<%=message %>
```

HTML encoding the output stops this attack in its tracks. **Figure 6** shows the main options for encoding the dangerous data.

These options prevent the kind of attack shown in the example and should be used in your applications.

> The AntiXSS Library has gone through a very nice rewrite.

It's important to know your controls. Which controls HTML encode your data for you and which controls don't? For example, a TextBox control does HTML encode the rendered output, but a LiteralControl doesn't. This is an important distinction. A textbox assigned:

```
yourTextBoxControl.Text = "Test <script>alert('bip')</script>";
```

correctly renders the text to the page as:

```
Test &lt;script&gt;alert(&#39;bip&#39;)&lt;/script&gt;
```

In contrast:

```
yourLiteralControl.Text = "Test <script>alert('bip')</script>";
```

causes a JavaScript alert to be displayed on the page, confirming XSS vulnerability. The fix here is simply:

```
yourLiteralControl.Text = Server.HtmlEncode(
  "Test <script>alert('bip')</script>");
```

This is a bit trickier when using data binding in Web Forms. Look at the following example:

```
<asp:Repeater ID="Repeater1" runat="server">
  <ItemTemplate>
    <asp:TextBox ID="txtYourField" Text='<%# Bind("YourField") %>'
      runat="server"></asp:TextBox>
  </ItemTemplate>
</asp:Repeater>
```

Is this vulnerable? No, it's not. Even though the inline code seems as if it could write out the script or break out of the control quotes, it's indeed encoded.

Figure 6 **HTML-Encoding Options**

ASP.NET (Either MVC or Web Forms)	`<%=Server.HtmlEncode(message) %>`
Web Forms (ASP.NET 4 syntax)	`<%: message %>`
ASP.NET MVC 3 Razor	`@message`
Data Binding	Unfortunately, the data-binding syntax doesn't yet contain a built-in encoding syntax; it's coming in the next version of ASP.NET as `<%#: %>`. Until then, use: `<%# Server.HtmlEncode(Eval("PropertyName")) %>`
Better Encoding	From the AntiXSS Library in the Microsoft.Security.Application namespace: `Encoder.HtmlEncode(message)`

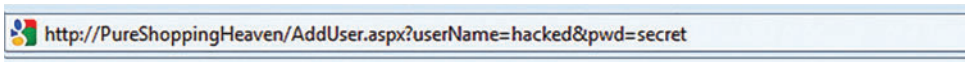


Figure 7 **Passing Information on the URL**

What about this:

```
<asp:Repeater ID="Repeater2" runat="server">
  <ItemTemplate>
    <%# Eval("YourField") %>
  </ItemTemplate>
</asp:Repeater>
```

Is it vulnerable? Yes, it is. The data-binding syntax <%# %> doesn't HTML encode. Here's the fix:

```
<asp:Repeater ID="Repeater2" runat="server">
  <ItemTemplate>
    <%#Server.HtmlEncode((string)Eval("YourText"))%>
  </ItemTemplate>
</asp:Repeater>
```

Be aware that if you use Bind in this scenario you can't wrap a Server.HtmlEncode around it because of how Bind compiles behind the scenes as two separate calls. This will fail:

```
<asp:Repeater ID="Repeater2" runat="server">
  <ItemTemplate>
    <%#Server.HtmlEncode((string)Bind("YourText"))%>
  </ItemTemplate>
</asp:Repeater>
```

If you use Bind and aren't assigning the text to a control that HTML encodes (such as the TextBox control), consider using Eval instead so you can wrap the call to Server.HtmlEncode as in the previous example.

The same concept of data binding doesn't exist in ASP.NET MVC, so you need to know if the HTML helpers will encode. The helpers for labels and textboxes do HTML encode. For example, this code:

```
@Html.TextBox("customerName", "<script>alert('bip')</script>")
@Html.Label("<script>alert('bip')</script>")
```

renders as:

```
<input id="customerName" name="customerName" type="text"
  value="&lt;script&gt;alert(&#39;bip&#39;)&lt;/script&gt;" />
<label for="">&lt;script&gt;alert(&#39;bip&#39;)&lt;/script&gt;</label>
```

I mentioned the AntiXSS earlier. Currently at version 4.1 beta 1, the AntiXSS Library has gone through a very nice rewrite and, as far as security is concerned, provides a better HTML encoder than the one that comes with ASP.NET. It's not that there's anything wrong with Server.HtmlEncode, but its focus is compatibility, not security. AntiXSS uses a different approach to encode. You can read more about it at msdn.microsoft.com/security/aa973814.

The beta is available at bit.ly/gMcB5K. Do check to see if AntiXSS is out of beta yet. If not, you'll need to download the code and compile it. Jon Galloway has an excellent post on this at bit.ly/IGpKWX.

To use the AntiXSS encoder, you can simply make the following call:

```
<%@ Import Namespace="Microsoft.Security.Application" %>
...
...
<%= Encoder.HtmlEncode(plainText)%>
```

ASP.NET MVC 4 added a great new feature that lets you override the default ASP HTML encoder, and you can use the AntiXSS encoder in its place. As of this writing, you need version 4.1; because it's currently in beta, you must download the code, compile it and add the library as a reference to your application—which takes all of five minutes. Then, in your web.config, add the following line in the <system.web> section:

```
<httpRuntime encoderType=
  "Microsoft.Security.Application.AntiXssEncoder, AntiXssLibrary"/>
```

Now, any HTML-encoding call made through any of the syntaxes listed in **Figure 6**, including the ASP.NET MVC 3 Razor syntax, will get encoded by the AntiXSS library. How's that for a pluggable feature?

This library also includes a Sanitizer object that can be used to cleanse HTML before storing it to a database, which is very useful if you provide a WYSIWYG editor to the user for editing HTML. This call attempts to remove script from the string:

```
using Microsoft.Security.Application;
...
...
string wysiwygData = "before <script>alert('bip ')</script> after ";
string cleanData = Sanitizer.GetSafeHtmlFragment(wysiwygData);
```

This results in the following cleaned string that can then be saved to the database:

```
cleanData = "before  after ";
```

ASP.NET MVC 4 added a great new feature that lets you override the default ASP HTML encoder.

Cross-Site Request Forgery (CSRF)

What Is It? Cross-site request forgery, or CSRF (pronounced *sea-surf*), is an attack that occurs when someone takes advantage of the trust between your browser and a Web site to execute a command using the innocent user's session. This attack is a bit more difficult to imagine without seeing the details, so let's get right to it.

How Is It Exploited? Suppose John is authenticated as an administrator on the PureShoppingHeaven site. PureShoppingHeaven has a URL that's restricted to admin access and allows information to be passed on the URL to execute an action, such as creating a new user, as shown in **Figure 7**.

Figure 8 **Preventing a One-Click Attack**

```
void Page_Init(object sender, EventArgs e)
{
  if (Session.IsNewSession)
  {
    // Force session to be created;
    // otherwise the session ID changes on every request.
    Session["ForceSession"] = DateTime.Now;
  }

  // 'Sign' the viewstate with the current session.
  this.ViewStateUserKey = Session.SessionID;


  if (Page.EnableViewState)
  {
    // Make sure ViewState wasn't passed on the querystring.
    // This helps prevent one-click attacks.
    if (!string.IsNullOrEmpty(Request.Params["__VIEWSTATE"]) &&
      string.IsNullOrEmpty(Request.Form["__VIEWSTATE"]))
    {
      throw new Exception("Viewstate existed, but not on the form.");
    }
  }


}
```

If an attacker can get John to request this URL through any of a variety of methods, his browser will request it from the server and send over whatever authentication information might already be cached or in use in John's browser, such as authentication cookies or other authentication tokens, including Windows Authentication.

This is a simple example, but CSRF attacks can be far more sophisticated and can incorporate form POSTs in addition to GET requests and can take advantage of other attacks such as XSS at the same time.

> By design the browser will send over any valid cookies or authentication information.

Suppose John visits a vulnerable social networking site that has been exploited. Perhaps an attacker has placed a bit of JavaScript on the page via an XSS vulnerability that now requests the AddUser.aspx URL under John's session. This dump from Fiddler (fiddler2.com) after John visits the Web page shows the browser also is sending over a custom site-authentication cookie:

```
GET http://pureshoppingheaven/AddUser.aspx?userName=hacked&pwd=secret HTTP/1.1
Host: pureshoppingheaven
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)
Cookie: CUSTOMERAUTHCOOKIE=a465bc0b-e1e2-4052-8292-484d884229ab
```

This all happens without John knowing it. What's important to understand is that by design the browser will send over any valid cookies or authentication information. Have you ever noticed that your e-mail client generally doesn't load images by default? One reason is to prevent CSRF. If you received an HTML-formatted e-mail with an embedded image tag such as the following, that URL would be requested and the server would execute that action if you're authenticated to that Web site:

```
<img src='yoursite/createuser.aspx?id=hacked&pwd=hacked' />
```

If you happened to be an admin on "yoursite" who's already authenticated, the browser would happily send over that GET request along with any credentials. The server sees this as a valid request by an authenticated user, and it will execute that request without you ever knowing, because there's no valid image response to be rendered in your e-mail client.

How Do You Prevent CSRF? To prevent CSRF, you start by following certain rules:

1. Ensure that a request can't be replayed by simply clicking on a link for a GET request. The HTTP spec for GET requests implies GET requests should only be used for retrieval, not state modification.
2. Ensure that a request can't be replayed if an attacker has used JavaScript to mimic a form POST request.
3. Prevent any actions via GET. For example, don't allow records to be created or deleted via a URL. Ideally, these should require some user interaction. While this doesn't prevent a smarter form-based attack, it limits a host of easier attacks, such as the kind described in the e-mail image example as well as basic links embedded in XSS-compromised sites.

Preventing attacks via Web Forms is handled a bit differently from ASP.NET MVC. With Web Forms, the ViewState MAC attribute can be signed, which helps protect against forgery as long as you don't set EnableViewStateMac=false. You also want to sign your ViewState with the current user session and prevent the ViewState from being passed in on the query string to block what some refer to as a one-click attack (see **Figure 8**).

The reason I assign a random session value here is to make sure the session is established. You could use any temporary session identifier, but the ASP.NET session ID will change on every single request until you actually create a session. You can't have the session ID changing with every request here, so you have to pin it down by creating the new session.

ASP.NET MVC contains its own set of built-in helpers that protect against CSRF using unique tokens passed in with the request. The helpers use not only a required hidden form field but also a cookie value, making it quite a bit more difficult to forge a request. These protections are easy to implement and absolutely essential to incorporate into your applications. To add @Html.AntiForgeryToken() inside <form> in your view, do this:

```
@using (Html.BeginForm())
{
  @Html.AntiForgeryToken();
  @Html.EditorForModel();
  <input type="submit" value="Submit" />
}
```

Decorate any controllers that accept post data with the [ValidateAntiForgeryToken], like so:

```
[HttpPost]
[ValidateAntiForgeryToken()]
public ActionResult Index(User user)
{
  ...
}
```

Understanding Vulnerability

This article looked at cross-site scripting and cross-site request forgery, two common ways Web applications are hacked. Added to the two covered last month, SQL injection and parameter tampering, you now have a good understanding of how applications can be vulnerable.

You've also seen how easy it is to build security into your applications to protect against some of the most common attacks. If you've already been building security into your software development lifecycle, great! If you haven't, there's no better time to start than now. You can audit your existing applications on a per-page/per-module basis and in most cases refactor them very easily. And protect your applications with SSL to prevent sniffing of your credentials. Remember to think about security before, during and after development. ■

Adam Tuliper *is a software architect with Cegedim and has been developing software for more than 20 years. He's a national INETA Community Speaker, and regularly speaks at conferences and .NET user groups. Check him out on Twitter at twitter.com/AdamTuliper, on his blog at completedevelopment.blogspot.com or at the new secure-coding.com Web site. For more in-depth information on hack-proofing your ASP.NET applications, see his upcoming Pluralsight video series.*

Thanks to the following technical expert for reviewing this article: *Barry Dorrans*

nsoftware

We didn't invent the Internet...

...but our components help **you** power the apps that bring it to business.

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**
  AS2, EDI/X12, NAESB, OFTP …
- **Credit Card Processing**
  Authorize.Net, TSYS, FDMS …
- **Shipping & Tracking**
  FedEx, UPS, USPS …
- **Accounting & Banking**
  QuickBooks, OFX …
- **Internet Business**
  Amazon, eBay, PayPal …
- **Internet Protocols**
  FTP, SMTP, IMAP, POP, WebDav …
- **Secure Connectivity**
  SSH, SFTP, SSL , Certificates …
- **Secure Email**
  S/MIME, OpenPGP …
- **Network Management**
  SNMP, MIB, LDAP, Monitoring …
- **Compression & Encryption**
  Zip, Gzip, Jar, AES …

Microsoft .NET | Microsoft BizTalk Server | Microsoft SQL Server | Windows Mobile | activex | Microsoft SharePoint 2010 | CodeGear Delphi | CodeGear C++Builder | Java | php

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity
powered by

To learn more please visit our website → **www.nsoftware.com**

Microsoft .NET, | Java | ActiveX | C++ | Delphi | C++ Builder | PHP | Objective-C | Windows | Windows Mobile | Mac OS X | Linux/Unix | iPhone

Customized On-Screen Keyboards with the .NET Framework

Christopher M. Frenz

The need to develop applications that require screen-based inputs has grown steadily in recent years. Developers have traditionally used on-screen data input to let individuals who couldn't use a computer keyboard enter data or make choices, especially in kiosk applications.

The rapid growth of tablet and mobile computing devices makes on-screen input more pervasive and useful than ever. One need not look past a local coffee shop or mode of public transportation to find people engrossed in the latest "app" for their mobile phone or tablet device. Given the expectations for continued growth in the mobile computing market, it's not hard to imagine that an understanding of on-screen input methodologies would be of great benefit to any developer.

Moreover, in addition to the mobile market, on-screen keyboards are also getting more popular in applications where security is critical, because screen-based input devices can help to prevent user information from being stolen using hardware-based keyloggers. For example, online stock brokerages such as TradeKing require users to enter all passwords using an on-screen keyboard to help boost security. But even though on-screen keyboards can help to improve security, they do have their own potential risks to consider. While on-screen keyboards mitigate the risk of any hardware-based keyloggers, they're still potentially susceptible to software-based input logging approaches as well as the much more common technique of "shoulder surfing" to view the input buttons a user presses. For example, one application uses captured video of a person typing into an iPad to identify which keys were typed by aligning the location of the pressed keys with an image of the iPad keyboard (see onforb.es/oobLp2).

Some techniques mitigate the effectiveness of shoulder surfing. One of the most common is a constant randomization of keys to prevent the mapping of captured keypress coordinates to any particular key. In this article, I'll create an on-screen keyboard that uses key randomization using the Microsoft .NET Framework. The application will allow the on-screen keyboard to be linked to the particular application for which on-screen input is desirable. However, this article describes the techniques needed to build an on-screen keyboard, and the sample application is designed to illustrate these techniques, not to provide a fully featured keyboard application.

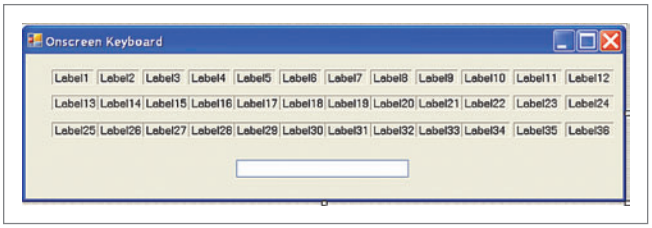


Figure 1 **The Design View of the On-Screen Keyboard GUI**

This article discusses:

- Laying out the GUI
- Using labels instead of buttons to avoid OS focus
- Adding keyboard functionality

Technologies discussed:

Microsoft .NET Framework, Visual Basic .NET

Code download available at:

code.msdn.microsoft.com/mag201201Keyboard

Figure 2 **Assigning Random, Unique Keys to Each Label Control**

```
Private Sub AssignKeys()
  Dim Character() As Char =
    {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", _
     "Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P", _
     "A", "S", "D", "F", "G", "H", "J", "K", "L", "Z", _
     "X", "C", "V", "B", "N", "M"}
  Dim Keys(36) As Char
  Dim I, X As Integer
  Dim Rand As New Random()
  Dim Used(36) As Integer
  Dim Unique As Boolean = False
    Used(0) = -1
    For I = 0 To 35
      Unique = False
      X = Rand.Next(0, 36)
      If Character(X) <> " " Then
        Keys(I) = Character(X)
        Character(X) = " "
      Else
        Do Until Unique = True
          X = Rand.Next(0, 36)
          If Character(X) <> " " Then
            Keys(I) = Character(X)
            Character(X) = " "
            Unique = True
          End If
        Loop
        End If
  Next
    Label1.Text = Keys(0)
    Label2.Text = Keys(1)
    Label3.Text = Keys(2)
    Label4.Text = Keys(3)
    Label5.Text = Keys(4)
    Label6.Text = Keys(5)
    Label7.Text = Keys(6)
    Label8.Text = Keys(7)
    Label9.Text = Keys(8)
    Label10.Text = Keys(9)
    Label11.Text = Keys(10)
    Label12.Text = Keys(11)
    Label13.Text = Keys(12)
    Label14.Text = Keys(13)
    Label15.Text = Keys(14)
    Label16.Text = Keys(15)
    Label17.Text = Keys(16)
    Label18.Text = Keys(17)
    Label19.Text = Keys(18)
    Label20.Text = Keys(19)
    Label21.Text = Keys(20)
    Label22.Text = Keys(21)
    Label23.Text = Keys(22)
    Label24.Text = Keys(23)
    Label25.Text = Keys(24)
    Label26.Text = Keys(25)
    Label27.Text = Keys(26)
    Label28.Text = Keys(27)
    Label29.Text = Keys(28)
    Label30.Text = Keys(29)
    Label31.Text = Keys(30)
    Label32.Text = Keys(31)
    Label33.Text = Keys(32)
    Label34.Text = Keys(33)
    Label35.Text = Keys(34)
    Label36.Text = Keys(35)
End Sub
```

The On-Screen Keyboard GUI

The first step is laying out the GUI, which includes a "key" for each letter of the alphabet and each number (see **Figure 1**). Before you start dragging controls onto your form, however, there are a few issues to consider. In most .NET applications (and other applications as well), whenever you want the user to submit input by clicking on something, the standard control to use would be a button. But you can't use a button control in an on-screen keyboard because on being clicked, a button control automatically gains the focus of the Windows OS. Because keyboard input is meant to go to the active foreground application (that is, the application with the focus), an on-screen keyboard should never gain the focus of the OS. Rather than use button controls for keys, I'll use label controls instead, because they support a Click event as buttons do, but unlike buttons aren't allowed to gain focus. Labels therefore make an ideal choice, being able to respond to clicks without initiating a change in application focus—with a bit of extra coding that you'll see in the next section. For purposes of the sample application, these labels will be named Label1 through Label36. A textbox, TextBox1, is also created to compare user inputs entered into the on-screen keyboard with those that appear in the external application (see **Figure 1**).

Staying Out of Focus

Although using a control such as a label is necessary to avoid causing the application to receive the focus of the OS, it's not enough in itself because the on-screen keyboard application can also receive the focus when the form itself is loaded and when the form or any control on the form is clicked by the mouse. To rectify this situation, I need to add some additional code to the keyboard application.

First, to prevent the form from gaining the focus when it's first loaded, I'll add the following code (I'm using Visual Basic .NET) to the application:

```
Private Const WS_EX_NOACTIVATE As Integer = &H8000000

Protected Overrides ReadOnly Property CreateParams() As CreateParams
  Get
    CreateParams = MyBase.CreateParams
    CreateParams.ExStyle = CreateParams.ExStyle And WS_EX_NOACTIVATE
    Return CreateParams
  End Get
End Property
```

This code overrides the form's CreateParams property, which is used in the creation of the form object. By overriding this property with the WS_EX_NOACTIVATE window style, I prevent the form from coming to the foreground upon loading, which means that loading the on-screen keyboard won't take the focus away from whatever other application was active at the time of launch. Once this code is put in place, it's important to next ensure that the application can't achieve focus via a mouse click. Adding this code accomplishes that:

```
Private Const WM_MOUSEACTIVATE As Integer = &H21
Private Const MA_NOACTIVATE As Integer = &H3
Protected Overrides Sub WndProc(ByRef m As Message)
  If (m.Msg = WM_MOUSEACTIVATE) Then
     m.Result = MA_NOACTIVATE
    Else
      MyBase.WndProc(m)
    End If
End Sub
```

This code overrides the form's WndProc function, which the form uses to receive all user input. The overriding function intercepts WM_MOUSEACTIVATE messages, which are sent when an inactive window is clicked on. It also ensures that the on-screen keyboard application doesn't gain the focus as a result of the mouse click by setting the function return value to MA_NOACTIVATE.

Figure 3 **Using the user32.dll API for Label_Click Events**

```
Private Sub Label1_Click(ByVal sender As System.Object,_
  ByVal e As System.EventArgs) Handles Label1.Click
  Dim X As Char
  X = CChar(Label1.Text)
  Dim theHandle As IntPtr
  theHandle = FindWindow(Nothing, "Untitled - Notepad")
  If theHandle <> IntPtr.Zero Then
    SetForegroundWindow(theHandle)
    SendKeys.Send(X)
  End If
  TextBox1.Text = TextBox1.Text & Label1.Text
  AssignKeys()
End Sub
```

Figure 4 **A New Instance of the On-Screen Keyboard**

The "Else" condition of this code ensures that all other mouse input messages are passed through, thereby allowing the on-screen keyboard application to detect label clicks without ever gaining focus.

Adding the Keyboard Functionality

At this point, I have a GUI for the application and code to ensure that it remains focus-free. It's now time to add the actual keyboard functionality. The first bit of this functionality will create a subroutine (AssignKeys) to assign random but unique keys to each label (see **Figure 2**).

The routine in **Figure 2** creates an array (Character) that contains all of the alphanumeric characters selected to appear on the keyboard and then applies the random number generator to select a random element from this array. As long as the element hasn't been previously selected, the character stored in that particular element of the array is copied to an array called Keys. This process repeats until all 36 characters are assigned to the Keys array, which randomizes the location of each character in the array. Once the array is randomized, the elements of the Keys array are assigned to the Text property of each Label to allow their assigned characters to be displayed on the screen. This AssignKeys subroutine is initially called on the execution of the Form_Load event.

Now that the characters have been assigned to their respective keys, I need to add code that handles converting the mouse clicks on the on-screen keyboard application into the equivalent of sending a keypress to the target application. To do this, I need to make use of the user32.dll API, which Windows uses to handle many UI-related functions such as window handling and other window management functions. To set the application up to properly make use of this API functionality, I'll add the following DLL Import statements to the Form class:

```
<DllImport("user32.dll", SetLastError:=True)> Private Shared Function
FindWindow(ByVal lpClassName As String, ByVal lpWindowName As String) As IntPtr
  End Function
<DllImport("user32.dll", SetLastError:=True)> Private Shared Function
SetForegroundWindow(ByVal hWnd As IntPtr) As Boolean
  End Function
```

The user32.dll API will be used every time a Label_Click event is launched, because each such event will resemble the code in **Figure 3**.

When a label click event arises, the variable "theHandle" is used to store the application handle for the application to which the on-screen keyboard is going to send its input. In this case, the application handle was set to a freshly loaded copy of Notepad because it's universally available on all Windows systems. If the application handle is currently present on the system, the application bearing that handle (Notepad) moves to the foreground and the character assigned to that label is sent to the application. The character is also appended to any text found within the keyboard application's textbox to demonstrate that the characters that appear in Notepad are the same characters received by the keyboard application itself. As a last step, the AssignKeys subroutine is called again to re-randomize the key positions and make shoulder surfing even more difficult. This procedure is illustrated in **Figure 4** and **Figure 5**, where **Figure 4** shows a newly loaded version of the application and **Figure 5** demonstrates the on-screen keyboard and Notepad after several keypresses on the on-screen keyboard.

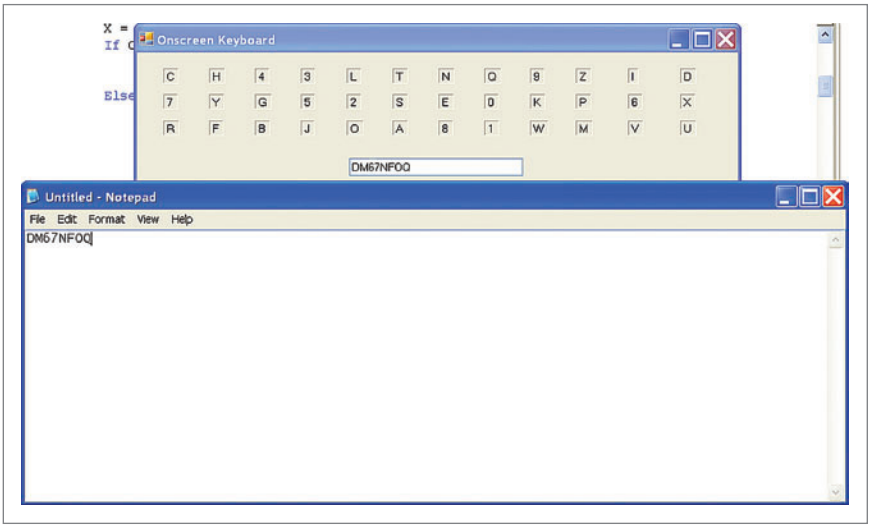


Figure 5: **The On-Screen Keyboard Sending Input to Notepad**

Enhanced Security and Mobile Porting

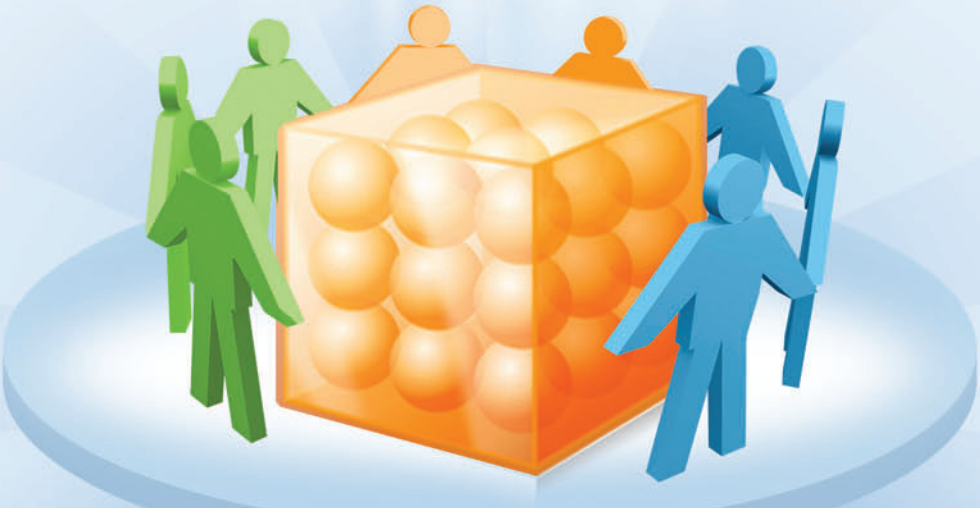
This article demonstrated the development of an on-screen keyboard using the .NET Framework. I hope it provided some insight into how on-screen keyboards can be developed for use in improving the security of certain elements of data entry or for use in porting .NET applications to mobile platforms. ■

CHRISTOPHER M. FRENZ *is the author of the programming books, "Visual Basic and Visual Basic .NET for Scientists and Engineers" (Apress, 2002) and "Pro Perl Parsing" (Apress, 2005). He can be reached at cfrenz@gmail.com.*

THANKS to the following technical expert for reviewing this article: *Robert Green*

InstallAware

NEW INSTALLAWARE 2012



App-V for the Masses

Visit www.installaware.com/landing/msdn.asp for this special pricing

SPECIAL

Instantly Build
App-V Packages

for Only
$890

Windows 8 Ultimate

NOW WITH SUPPORT FOR WINDOWS 8

App-V Builder:
Compiles any existing InstallAware project as an App-V Package

App-V Viewer:
Opens and inspects the contents of any pre-existing App-V Package

Hybrid 32 bit & 64 bit:
Combines both 32 bit and 64 bit applications inside a single App-V

MSI Deployment:
Creates an MSI file to silently push your App-V Packages

Command Line Support:
Automates your App-V Build process through the command line

Extending SSRS: Developing Custom Charting Components and Rendering Extensions

Manpreet Singh

SQL Server Reporting Services (SSRS) provides excellent charting capabilities that allow you to represent data and statistics visually in the form of both conventional and unconventional charts, and a variety of report-rendering extensions let you save a report in a variety of different formats such as PDF, CSV, XML, Word and Excel. Still, the native options may not always satisfy your business requirements or render a report exactly the way you want. Fortunately, you can extend the reporting services to create your own custom charting components and rendering extensions using the various extensibility features SSRS provides. In this article, I'll give you an overview of the process of creating a custom charting component and the various ways to integrate it with an SSRS report. I'll also describe how you can develop a custom report renderer (by extending a native one) to render a report just as you'd like. To try this yourself, download the complete code at code.msdn.microsoft.com/mag201201SSRS and use it as your starting point. Note that the process of creating a rendering extension and report items hasn't changed much since SQL Server 2005. Although I built the code samples using a SQL Server 2008 R2 environment, the concepts discussed in this article are very much applicable to SQL Server 2005 and 2008 as well. If you're developing for SQL Server 2008 R2 Reporting Services, you can build custom components using the Microsoft .NET Framework 4 as well. For SQL Server 2008 Reporting Services, however, the .NET Framework 3.5 is the highest version of the .NET Framework supported.

This article discusses:

- Extensibility in SQL Server Reporting Services
- Creating custom charting components
- Creating a custom report-rendering extension by extending a native one
- The WebSite Users Report and the WebSite Users Charting Component

Technologies discussed:

SQL Server, Microsoft .NET Framework 3.5

Code download available at:

code.msdn.microsoft.com/mag201201SSRS

The Web Site Users Report

To appreciate the process of developing a custom charting component and rendering extension, consider a scenario where an organization wants to generate a monthly Web Site Usage Report to illustrate the popularity of its, say, e-commerce Web site in different regions. The heart of the report is a Web Site Users Chart, which graphically represents the geographic distribution of users per region, as depicted in **Figure 1**.

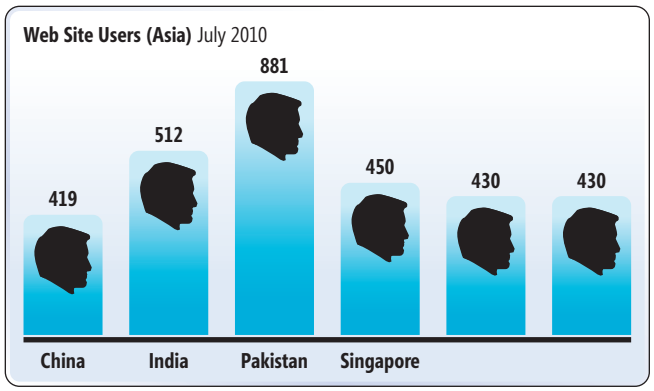


Figure 1 **Web Site Users Chart**

Although the chart is quite similar to a bar graph, none of the native charting components fit the bill because you can't replace the bars of the chart with a custom image representing a person, like the one in **Figure 1**.

Such a scenario definitely calls for creating your own custom charting component. Using the GDI+ base class libraries (part of the .NET Framework) and a little mathematics, it's easy to draw the chart using basic shapes like circles and rectangles and generate the chart in the form of a regular bitmap image. The output from such a component, which generates an image by using the .NET Framework graphics libraries, can be integrated with the SSRS report in a number of ways:

- Using Custom Report Items: SSRS supports the creation of custom report items, such as built-in charting components, that can be integrated into a report. The custom report items are rendered as images. The article, "Jazz Up Your Data Using Custom Report Items in SQL Server Reporting Services" (msdn.microsoft.com/magazine/cc188686), demonstrates how to create a custom report item.
- Using a SQL CLR Function: You can also use a SQL CLR user-defined function to integrate a GDI+ based charting component with SSRS. This requires loading system.drawing.dll into SQL Server. Once the core assembly for chart generation is loaded into SQL Server, you can create a user-defined function based on it. The image reporting item can then be configured to render the chart image from a database.
- Using an ASP.NET Handler: You can use an ASP.NET handler to integrate the charting component with both an SSRS report and a Web application. An ASP.NET handler can return the charting component as a downloadable bitmap image, with any parameters being passed to it using a query string. To integrate the handler output with SSRS, all you need to do is set the image source of the picture box to external and provide the URL of the handler as an expression.

The code samples accompanying this article demonstrate the process of creating the Web Site Users Chart and integrating it with the Web Site Usage Report using a SQL CLR function. The WebSiteUsersChartCore project contains the WebSiteUsersChart class, which renders the chart in the form of a JPEG image based on the monthly usage data returned by the dbo.usp_GetUsageData stored procedure. The WebSiteUsersReportDB.bak file contains a backup of the WebSiteUsersReportDB database with the necessary back-end tables and stored procedures. The SQL CLR function GetUsersChart is responsible for providing the necessary parameters to the charting component and getting the image output. The dataset DataSetWebSiteUsageChart of the RegionWiseReport gets the image from the SQL CLR function using the following SQL query, by passing in the required parameters:

```
SELECT dbo.usp_GetUsersChart(500, 300, N'Website Users', @RegionName,
  @DateID % 100, @DateID / 100) AS Chart
```

Finally, the image control ImageChart renders the image coming from the database.

The main advantage of using the SQL CLR function or the ASP.NET handler approach instead of developing a custom report item is that you can then use the same charting component outside of SSRS, say in a Web application, without the need for any further development effort or customization.

Custom Word Rendering Extension

Although the native Word rendering extension renders the Web Site Usage Report perfectly, some features, such as a table of contents or having some pages in portrait and others in landscape orientation, are not supported.

Writing a custom rendering extension from scratch isn't a trivial task; you want it to be capable of taking various combinations of report elements and converting them to the corresponding formatting and data elements supported by the target format. Moreover, extending the native SSRS rendering extensions is not supported directly. Luckily, the output from the native Word rendering extension meets most of my needs for the report, so all I really want is a way to add a table of contents to it and to be able to control the orientation at the page level.

To accomplish my goals, I can write a custom rendering extension, which gets the report output rendered by the native Word extension, then use Word automation to modify the report output in the desired way and stream the final output to the user.

The class CustomWordRenderer in the project CustomWordRenderingExtension in the source code highlights the important steps involved in achieving the desired effect. The output from the native Word rendering extension is merged with a standard header template that contains a title page and a table of contents, and can also contain items such as a copyright note, a disclaimer and so forth. The document map labels, which are rendered as Table Entries by the native extensions, have appropriate heading styles applied to make them appear in the table of contents. Finally, the table of contents is updated and the merged document is streamed to the user.

All the reporting services extensions must implement the IRenderingExtension and IExtension interfaces. These interfaces require you to provide implementation for the following methods and properties:

Members from the IExtension interface:

- LocalizedName property
- SetConfiguration function

Members from the IRenderingExtension interface:

- GetRenderingResource function
- Render function
- RenderStream function

Of all these members, the Render method is most important and contains the core implementation for the Custom Word Renderer.

Because I already get the formatted report output from the native Word rendering extension, I don't have to parse the various report elements, and hence the render method won't contain any code to deal directly with report data or formatting elements. All I need to do is get the report from the native Word rendering extension, open it with Microsoft Word (using Word automation), perform the required modifications and stream modified document to the user.

For more information on the process of developing custom rendering extensions, see the article, "Display Your Data Your Way with Custom Renderers for Reporting Services," at msdn.microsoft.com/magazine/cc163840.

An important point to note before you decide to make use of concepts discussed here is that Microsoft doesn't support server-side automation of Microsoft Office. If you do consider developing a custom rendering extension based on the approach I discussed, first take a look at the article, "Considerations for Server-Side Automation of Office," at support.microsoft.com/kb/257757.

If you don't want to use Office automation, there's another interesting approach you can consider—rendering the report as an XML file using the native XML rendering extension and applying a custom XSLT stylesheet to generate a WordML document. The disadvantage of this approach is that you'll end up storing report formatting in two places: one in the XSLT stylesheet and other in the report RDL file.

Using the Sample Code

The source code for this article contains the following artifacts:

- WebSiteUsersChartCore project
- TestHarness project
- CustomWordRenderingExtension project
- WebSiteUsersReport RDL file
- RegionWiseReport RDL file
- WebSiteUsersReportDB database backup file
- Report header template file

The WebSiteUsersChartCore project demonstrates the process of creating the WebSite Users Charting component from region-wise usage statistics. The TestHarness project is used to test and verify the output of WebSiteUsersChartCore assembly by rendering the image generated by the charting component using a Windows Form application. The CustomWordRenderingExtension project contains the implementation for the custom Word rendering extension, based on the Word automation approach. The WebSiteUsersReport is the primary report that calls the RegionWiseReport subreport for every region, to render the region-wise contents. The RegionWiseReport report also calls the WebSite Users Charting component to render the WebSite Users Chart. It does so using the image control via a call to the usp_GetUsersChart SQL CLR scalar-valued function by passing in the appropriate parameters. The WebsiteUsersReportDB.bak file contains the backup for the WebSiteUsersReportDB database. The reports and the charting components are based on the data from this database. The header template file contains a title page and a table of contents, in portrait format. This file is appended to the beginning of the native Word document report, using the Word automation.

To deploy the sample code, build the WebSiteUsersChart solution containing the WebSiteUsersChartCore and CustomWordRenderingExtension projects. Copy the CustomWordRenderingExtension.dll assembly to the report server's bin directory. Make the following entry in your report server's rssrvpolicy.config file to grant full trust to the custom rendering extension:

```
<CodeGroup
class="UnionCodeGroup" version="1"
PermissionSetName="FullTrust"
Name="CUSTOM WORD"
Description="This code group grants Custom Word Renderer code full trust.">
  <IMembershipCondition
    class="UrlMembershipCondition"
    version="1"
    Url="C:\Program Files\Microsoft SQL Server\MSRS10_50.MSSQLSERVER\Reporting
        Services\ReportServer\bin\CustomWordRenderingExtension.dll" />
</CodeGroup>
```

Next, make the following entry in the rsreportserver.config file to register the rendering extension with the report server:

```
<Extension Name="Custom WORD" Type=
  "CustomWordRenderingExtension.CustomWordRenderer,
  CustomWordRenderingExtension">
  <Configuration>
    <DeviceInfo>
    <SourceHeaderFileName>C:\WorkingDirectory\
      Header.doc</SourceHeaderFileName>
    <SourceBodyFileName>C:\WorkingDirectory\Body.doc</SourceBodyFileName>
    <MergedFileName>C:\WorkingDirectory\MergedOutput.doc</MergedFileName>
    </DeviceInfo>
  </Configuration>
</Extension>
```

Create a working directory on the C: drive for the rendering extension and copy Header.doc to it. If you want to use a different directory, don't forget to make the appropriate changes in the rsreportserver.config file as well; these configuration entries are picked up by the custom Word renderer. Of course, you must have Microsoft Word installed for the rendering extension to work.

Next, deploy the WebSiteUsersReport and RegionWiseReport reports on the report server. Run the WebSiteUsersReport report. Click on the Export menu and examine the contents of the drop-down—you should see the "CUSTOM WORD" rendering extension in the list. Go ahead and do an export. If you end up getting an empty document, examine the event log for errors.

Wrapping Up

Whenever native charting components don't fit the bill, you should consider implementing a custom component. You can develop complex charts by leveraging your GDI+ development skill. Keep in mind that you can write custom renderers without having to write everything from scratch, by modifying the output of the native ones.

While it's always possible to go completely custom and build new applications for report generation, it's often very easy to achieve the same effect using SSRS with a few pieces of custom code plugged in at the right places. So go ahead and build your own charting components and report renderers without fear. ■

MANPREET SINGH *is a consultant with Microsoft Services Global Delivery, where he's a part of the Business Intelligence and Integration Engineering group. He works primarily on design and development of .NET-centric Business Intelligence solutions based on the Microsoft Business Intelligence stack.*

THANKS to the following technical expert for reviewing this article: *Yaswant Vishwakarma*

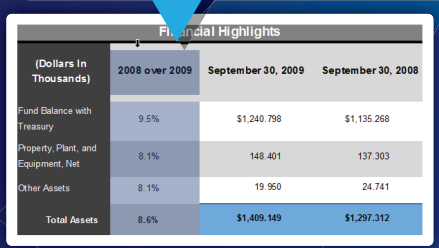
WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

WORD PROCESSING COMPONENTS

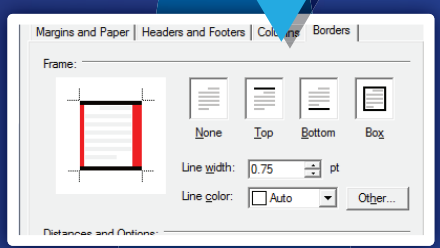
VERSION 17.0 RELEASED

NEW SPELL CHECK COMPONENT

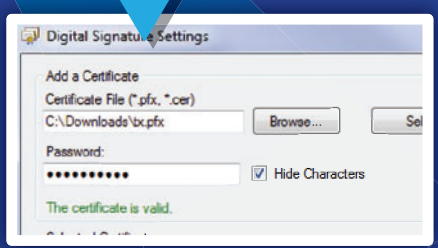
CELL MERGING, COL SELECTION



PAGE BORDERS



DIGITAL SIGNATURES IN PDF



TX TEXT CONTROL® word processing components

Microsoft Visual Studio PARTNER

Word Processing Components for Windows Forms, WPF & ASP.NET

WWW.TEXTCONTROL.COM

US +1 877 - 462 - 4772 (toll-free)
EU +49 421 - 4270671 - 0

JAMES MCCAFFREY

Simulated Annealing and Testing

In this month's column I present C# code that implements a Simulated Annealing (SA) algorithm to solve a scheduling problem. An SA algorithm is an artificial intelligence technique based on the behavior of cooling metal. It can be used to find solutions to difficult or impossible combinatorial optimization problems.

The best way to see where I'm headed is to take a look at **Figure 1** and **Figure 2**. The table in **Figure 1** describes a scheduling problem: five workers and six tasks. Each entry in the table represents how long it takes for a worker to complete a task. For example, worker w2 needs 5.5 hours to complete task t3. An empty entry in a row indicates that the corresponding worker can't perform a task. The problem is to assign each of the six tasks to one of the workers in such a way that the total time to complete all tasks is minimized. Additionally, we assume that every time a worker switches to a new task, there's a 3.5-hour retooling penalty.

> An SA algorithm is an artificial intelligence technique based on the behavior of cooling metal.

Figure 2 shows a program that uses an SA algorithm to find a solution to the scheduling problem. The program begins by generating a random solution. In SA terminology, a potential solution is called the state of the system. The initial state is [4 0 0 2 2 3], which means task 0 has been assigned to worker 4, task 1 has been assigned to worker 0, task 2 has been assigned to worker 0, task 3 has been assigned to worker 2, task 4 has been assigned to worker 2 and task 5 has been assigned to worker 3. The total time for the initial random state is 2.5 + 3.5 + 2.5 + 5.5 + 3.5 + 4.5 = 22 hours plus a 3.5-hour retooling penalty for worker 0 and a 3.5-hour penalty for worker 2, for a total of 29 hours. In SA terminology, the quantity you're trying to minimize (or less frequently maximize) is called the energy of the state.

The program enters a loop. In each iteration of the loop, the SA algorithm generates an adjacent state and evaluates that adjacent state to see if it's better than the current state. Behind the scenes, the SA algorithm uses a temperature variable that slowly decreases, as I'll explain shortly. The SA loop ends when the temperature cools sufficiently close to zero. The program concludes by displaying the best solution found.

Code download available at code.msdn.microsoft.com/mag201201TestRun.

Figure 1 **Time for Worker to Complete Task**

	t0	t1	t2	t3	t4	t5
w0	7.5	3.5	2.5			
w1		1.5	4.5	3.5		
w2			3.5	5.5	3.5	
w3				6.5	1.5	4.5
w4	2.5				2.5	2.5

This example is artificially simple because with six tasks where each task can be performed by one of three workers, there are only 3^6 possible combinations, which equals 729, so you could just evaluate every one. But suppose you had a problem with 20 tasks where each task can be performed by one of 12 workers. There would be 12^{20} combinations, which equals a whopping 3,833,759,992,447,475,122,176. Even if you could evaluate 1 million possible solutions per second, you'd need about 121 million years to evaluate every possibility.

SA is a metaheuristic—that is, a general conceptual framework that can be used to create a specific algorithm to solve a specific problem. It's best used for combinatorial optimization problems where there's no practical deterministic solution algorithm. First described in a 1983 paper by S. Kirkpatrick, C. Gelatt and M. Vecchi, SA is loosely based on the annealing behavior of cooling metal. When some metals are heated to a high temperature, the atoms and molecules break their bonds. When the metal is slowly cooled, the atoms and molecules reform their bonds in such a way that the energy of the system is minimized.

This column assumes you have intermediate-level programming skills. I implement the SA program using C#, but you shouldn't have too much trouble refactoring my code to a different language such as Visual Basic or Python. In the sections that follow, I'll walk you through the program that generated **Figure 2**. All of the code is available as a download from code.msdn.microsoft.com/mag201201TestRun. I think you'll find the ability to understand and use SA an interesting and possibly useful addition to your personal skill set.

Program Structure

I implemented the SA demo program as a single C# console application. The overall structure of the program is shown in **Figure 3**.

I used Visual Studio to create a console application program named SimulatedAnnealing. In the Solution Explorer window,

I renamed the default Program.cs file to SimulatedAnnealingProgram.cs, which automatically renamed the single class in the project. I deleted all the template-generated using statements except for the System namespace—SA is quite simple and typically doesn't need much library support. I declared a class-scope Random object named random. SA algorithms are probabilistic, as you'll see shortly.

The heart of the SA algorithm processing is a while loop. The loop is controlled by a loop counter variable named "iteration" and by a variable that represents the temperature of the system. In practice, the temperature variable almost always reaches near-zero and terminates the loop before the loop counter reaches its maximum value and terminates the loop.

SA algorithms must have three problem-specific methods as suggested in **Figure 3**. The SA algorithm must have a method that generates an initial (usually random) state/solution. The SA algorithm must have a method that generates an adjacent state relative to a given state. And the SA algorithm must have a method that computes the energy/cost of a state—the value you're trying to minimize or maximize. In **Figure 3** these are methods RandomState, AdjacentState and Energy, respectively. Method AcceptanceProb generates a value used to determine if the current state of the system transitions to the adjacent state even when the adjacent state is worse than the current state. Method MakeProblemData is a helper and creates a data structure matrix that corresponds with **Figure 1**. The overloaded Display methods and the Interpret method are just helpers to display information.

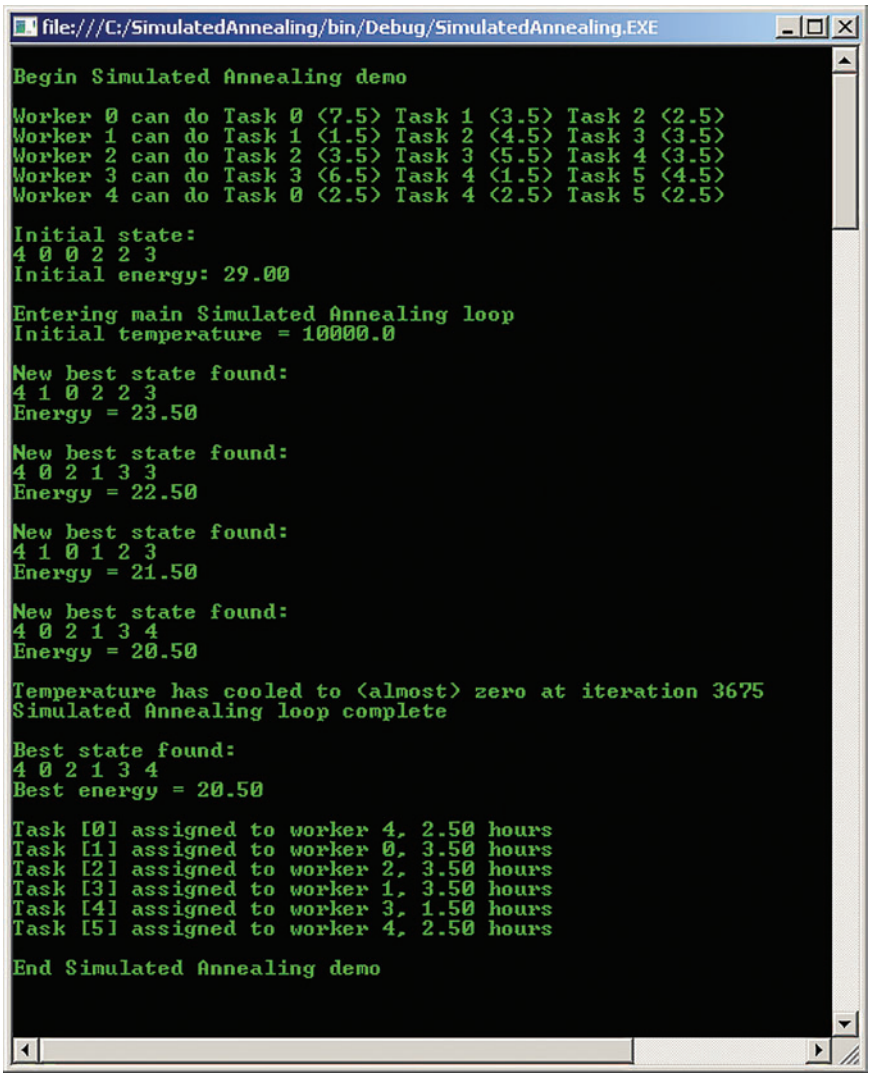


Figure 2 **SimulatedAnnealing in Action**

Program Initialization

The Main method begins like so:

```
try
{
  Console.WriteLine("\nBegin Simulated Annealing demo\n");
  Console.WriteLine("Worker 0 can do Task 0 (7.5) Task 1 (3.5) Task 2 (2.5)");
  Console.WriteLine("Worker 1 can do Task 1 (1.5) Task 2 (4.5) Task 3 (3.5)");
  Console.WriteLine("Worker 2 can do Task 2 (3.5) Task 3 (5.5) Task 4 (3.5)");
  Console.WriteLine("Worker 3 can do Task 3 (6.5) Task 4 (1.5) Task 5 (4.5)");
  Console.WriteLine("Worker 4 can do Task 0 (2.5) Task 4 (2.5) Task 5 (2.5)");
  ...
```

I wrap all SA code in a single, high-level try-catch block and display the dummy problem that I intend to set up. Here, I'm using an artificially simple example—but one that's representative of the kinds of combinatorial problems that are suited for a solution by SA algorithms. Next comes:

```
random = new Random(0);
int numWorkers = 5;
int numTasks = 6;
double[][] problemData = MakeProblemData(numWorkers, numTasks);
```

I initialize the Random object using an arbitrary seed value of 0. Then I call helper method MakeProblemData to construct the data structure shown in **Figure 1**. I'll present MakeProblemData and the other helper methods after I finish presenting all the code in the Main method. Next comes:

```
int[] state = RandomState(problemData);
double energy = Energy(state, problemData);
int[] bestState = state;
double bestEnergy = energy;
int[] adjState;
double adjEnergy;
```

> The heart of the SA algorithm processing is a while loop.

I call helper method RandomState to generate a random state/solution for the problem. State is represented as an int array where the array index represents a task and the value in the array at the index represents the worker assigned to the task. Helper method Energy

computes the total time required by its state parameter, taking into account the 3.5-hour penalty for retooling every time a worker does an additional task. I'll track the best state found and its corresponding energy, so I declare variables bestState and bestEngery. Variables adjState and adjEnergy are used to hold a state that's adjacent to the current state, and the corresponding energy. Next comes:

```
int iteration = 0;
int maxIteration = 1000000;
double currTemp = 10000.0;
double alpha = 0.995;
```

The primary SA processing loop terminates on one of two conditions: when a counter exceeds a maximum value or when the temperature variable decreases to a value close to zero. I name the loop counter "iteration," but I could've called it "counter" or "time" or "tick" or something similar. I name the temperature variable currTemp rather than temp so there's less chance someone reviewing the code might interpret it as a temporary variable. Variable alpha represents the cooling rate, or a factor that determines how the temperature variable decreases, or cools, each time through the processing loop. Next comes:

Figure 3 **SimulatedAnnealing Program Structure**

```
using System;
namespace SimulatedAnnealing
{
  class SimulatedAnnealingProgram
  {
    static Random random;

    static void Main(string[] args)
    {
      try
      {
        // Set up problem data.
        // Create random state.
        // Set up SA variables for temperature and cooling rate.

        while (iteration < maxIteration && currTemp > 0.0001)
        {
          // Create adjacent state.
          // Compute energy of adjacent state.
          // Check if adjacent state is new best.
          // If adjacent state better, accept state with varying probability.
          // Decrease temperature and increase iteration counter.
        }
        // Display best state found.
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
      }
    } // Main

    static double[][] MakeProblemData(int numWorkers, int numTasks) { ... }

    static int[] RandomState(double[][] problemData) { ... }

    static int[] AdjacentState(int[] currState,
      double[][] problemData) { ... }

    static double Energy(int[] state, double[][] problemData) { ... }

    static double AcceptanceProb(double energy, double adjEnergy,
      double currTemp) { ... }

    static void Display(double[][] matrix) { ... }

    static void Display(int[] vector) { ... }

    static void Interpret(int[] state, double[][] problemData) { ... }

  } // Program
} // ns
```

```
Console.WriteLine("\nInitial state:");
Display(state);
Console.WriteLine("Initial energy: " + energy.ToString("F2"));
Console.WriteLine("\nEntering main Simulated Annealing loop");
Console.WriteLine("Initial temperature = " + currTemp.ToString("F1") + "\n");
```

I'm using an artificially simple example—but one that's representative of the kinds of combinatorial problems that are suited for a solution by SA algorithms.

Before entering the main processing loop, I display some informational messages about the initial state, energy and temperature. You might want to display additional information such as the cooling rate. Here's the loop:

```
while (iteration < maxIteration && currTemp > 0.0001)
{
  adjState = AdjacentState(state, problemData);
  adjEnergy = Energy(adjState, problemData);

  if (adjEnergy < bestEnergy)
  {
    bestState = adjState;
    bestEnergy = adjEnergy;

    Console.WriteLine("New best state found:");
    Display(bestState);
    Console.WriteLine("Energy = " + bestEnergy.ToString("F2") + "\n");
  }
...
```

Notice the loop control exits when the temperature variable drops below 0.0001 rather than when it hits 0.0. You might want to parameterize that minimum temperature value. After creating an adjacent state and computing its energy, I check to see if that adjacent state is a new global best solution, and if so I save that information. I can copy the best state by reference because method AdjacentState allocates a new array, but I could've made an explicit copy. Whenever a new global best state is found, I display it and its energy. The main processing loop ends like this:

```
  double p = random.NextDouble();
  if (AcceptanceProb(energy, adjEnergy, currTemp) > p)
  {
    state = adjState;
    energy = adjEnergy;
  }

  currTemp = currTemp * alpha;
  ++iteration;
} // While
```

The loop finishes up by first generating a random value p greater than or equal to 0.0 and strictly less than 1.0 and comparing that value against the return from helper method AcceptanceProb. If the acceptance probability exceeds the random value, the current state transitions to the adjacent state. Next, the current temperature is decreased slightly by multiplying by the cooling factor, and the loop counter variable is incremented. Next comes:

Figure 4 **The Energy Method**

```
static double Energy(int[] state, double[][] problemData)
{
  double result = 0.0;
  for (int t = 0; t < state.Length; ++t) {
    int worker = state[t];
    double time = problemData[worker][t];
    result += time;
  }
  int numWorkers = problemData.Length;
  int[] numJobs = new int[numWorkers];
  for (int t = 0; t < state.Length; ++t) {
    int worker = state[t];
    ++numJobs[worker];
    if (numJobs[worker] > 1) result += 3.50;
  }
  return result;
}
```

```
Console.Write("Temperature has cooled to (almost) zero ");
Console.WriteLine("at iteration " + iteration);
Console.WriteLine("Simulated Annealing loop complete");
Console.WriteLine("\nBest state found:");
Display(bestState);
Console.WriteLine("Best energy = " + bestEnergy.ToString("F2") + "\n");
Interpret(bestState, problemData);
Console.WriteLine("\nEnd Simulated Annealing demo\n");
Console.ReadLine();
```

After the main SA processing loop completes, I display the best state (solution) found and its corresponding energy (hours). The Main method ends like this:

```
...
  } // Try
  catch (Exception ex)
  {
    Console.WriteLine(ex.Message);
    Console.ReadLine();
  }
} // Main
```

The method wraps up by handling any exceptions simply by displaying the exception's message.

The Helper Methods

The code for helper method MakeProblemData is:

```
static double[][] MakeProblemData(int numWorkers, int numTasks)
{
  double[][] result = new double[numWorkers][];
  for (int w = 0; w < result.Length; ++w)
    result[w] = new double[numTasks];

  result[0][0] = 7.5; result[0][1] = 3.5; result[0][2] = 2.5;
  result[1][1] = 1.5; result[1][2] = 4.5; result[1][3] = 3.5;
  result[2][2] = 3.5; result[2][3] = 5.5; result[2][4] = 3.5;
  result[3][3] = 6.5; result[3][4] = 1.5; result[3][5] = 4.5;
  result[4][0] = 2.5; result[4][4] = 2.5; result[4][5] = 2.5;
  return result;
}
```

I decided to use type double[][]—that is, an array of arrays—to hold my scheduling problem definition. The C# language, unlike many C-family languages, does support a built-in two-dimensional array, so I could've used type double[,] but an array of arrays is easier to refactor if you want to recode my example to a language that doesn't support two-dimensional arrays. In this example I arbitrarily put the worker index first and the task index second (so result[1][3] is the time required by worker 1 to perform task 3), but I could've reversed the order. Notice that C# automatically initializes type double array cells to 0.0, so I don't have to explicitly do so. I could've used some other value, such as -1.0 to indicate that a worker can't perform a particular task.

Helper method RandomState is:

```
static int[] RandomState(double[][] problemData)
{
  int numWorkers = problemData.Length;
  int numTasks = problemData[0].Length;
  int[] state = new int[numTasks];
  for (int t = 0; t < numTasks; ++t) {
    int w = random.Next(0, numWorkers);
    while (problemData[w][t] == 0.0) {
      ++w; if (w > numWorkers - 1) w = 0;
    }
    state[t] = w;
  }
  return state;
}
```

Recall that a state represents a solution and that a state is an int array where the index is the task and the value is the worker. For each cell in state, I generate a random worker w. But that worker might not be able to perform the task, so I check to see if the corresponding value in the problem data matrix is 0.0 (meaning the worker can't do the task), and if so I try the next worker, being careful to wrap back to worker 0 if I exceed the index of the last worker.

> By sometimes going to a worse state, you can escape non-optimal dead-end states.

Helper method AdjacentState is:

```
static int[] AdjacentState(int[] currState, double[][] problemData)
{
  int numWorkers = problemData.Length;
  int numTasks = problemData[0].Length;
  int[] state = new int[numTasks];
  int task = random.Next(0, numTasks);
  int worker = random.Next(0, numWorkers);
  while (problemData[worker][task] == 0.0) {
    ++worker; if (worker > numWorkers - 1) worker = 0;
  }
  currState.CopyTo(state, 0);
  state[task] = worker;
  return state;
}
```

Figure 5 **The Display and Interpret Helper Methods**

```
static void Display(double[][] matrix)
{
  for (int i = 0; i < matrix.Length; ++i) {
    for (int j = 0; j < matrix[i].Length; ++j)
      Console.Write(matrix[i][j].ToString("F2") + " ");
    Console.WriteLine("");
  }
}

static void Display(int[] vector)
{
  for (int i = 0; i < vector.Length; ++i)
    Console.Write(vector[i] + " ");
  Console.WriteLine("");
}

static void Interpret(int[] state, double[][] problemData)
{
  for (int t = 0; t < state.Length; ++t) { // task
    int w = state[t]; // worker
    Console.Write("Task [" + t + "] assigned to worker ");
    Console.WriteLine(w + ", " + problemData[w][t].ToString("F2"));
  }
}
```

Method AdjacentState starts with a given state, then selects a random task and then selects a random worker who can do that task. Note that this is a pretty crude approach; it doesn't check to see if the randomly generated new worker is the same as the current worker, so the return state might be the same as the current state. Depending on the nature of the problem being targeted by an SA algorithm, you might want to insert code logic that ensures an adjacent state is different from the current state.

The Energy method is shown in **Figure 4**.

> SA algorithms are simple to implement and can be powerful tools, but they do have weaknesses.

The Energy method first walks through each task in the state array, grabs the assigned worker value, looks up the time required in the problem data matrix, and accumulates the result. Next, the method counts the number of times a worker does more than one task and adds a 3.5-hour retooling penalty for every additional task. In this example, computing the energy of a state is quick; however, in most realistic SA algorithms, the Energy method dominates the running time, so you'll want to make sure the method is as efficient as possible.

Helper method AcceptanceProb is:

```
static double AcceptanceProb(double energy, double adjEnergy,
  double currTemp)
{
  if (adjEnergy < energy)
    return 1.0;
  else
    return Math.Exp((energy - adjEnergy) / currTemp);
}
```

If the energy of the adjacent state is less than (or more than, in the case of a maximization problem) the energy of the current state, the method returns 1.0, so the current state will always transition to the new, better adjacent state. But if the energy of the adjacent state is the same as or worse than the current state, the method returns a value less than 1.0, which depends on the current temperature. For high values of temperature early in the algorithm, the return value is close to 1.0, so the current state will often transition to the new, worse adjacent state. But as the temperature cools, the return value from AcceptanceProb becomes smaller and smaller, so there's less chance of transitioning to a worse state.

The idea here is that you sometimes—especially early in the algorithm—want to go to a worse state so you don't converge on a non-optimal local solution. By sometimes going to a worse state, you can escape non-optimal dead-end states. Notice that because the function performs arithmetic division by the value of the current temperature, the temperature can't be allowed to reach 0. The acceptance function used here is the most common function and is based on some underlying math assumptions, but there's no reason you can't use other acceptance functions.

The Display and Interpret Helper methods are extremely simple, as shown in **Figure 5**.

Some Weaknesses

SA algorithms are simple to implement and can be powerful tools, but they do have weaknesses. Because these algorithms are most often used in situations where there's no good deterministic solving algorithm, in general you won't know when, or even if, you hit an optimal solution. For example, in **Figure 1**, the best solution found had an energy of 20.5 hours, but by running the algorithm a bit longer you can find a state that has energy of 19.5 hours. So, when using SAs, you must be willing to accept a good but not necessarily optimal solution. A related weakness with SA algorithms and other algorithms based on the behavior of natural systems is that they require the specification of free parameters such as the initial temperature and the cooling rate. The effectiveness and performance of these algorithms, including SAs, are often quite sensitive to the values you select for the free parameters.

SA algorithms are closely related to Simulated Bee Colony (SBC) algorithms, which I described in the April 2011 issue (msdn.microsoft.com/magazine/gg983491). Both techniques are well suited for solving combinatorial, non-numeric optimization problems. In general, SAs are faster than SBCs, but SBCs tend to produce better solutions at the expense of performance.

> The use of artificial intelligence techniques in software testing is an area that's almost entirely unexplored.

The use of artificial intelligence techniques in software testing is an area that's almost entirely unexplored. One example where SAs might be used in software testing is as algorithm validation. Suppose you have some combinatorial problem that can in fact be solved using a deterministic algorithm. One example is the graph shortest-path problem, which can be solved by several efficient but relatively complicated algorithms such as Dijkstra's algorithm. If you've coded a shortest-path algorithm, you could test it by quickly coding up a simple SA algorithm and comparing results. If the SA result is better than the deterministic algorithm, you know you have a bug in your deterministic algorithm. ■

Dr. James McCaffrey *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He's worked on several Microsoft products, including Internet Explorer and MSN Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006), and can be reached at jammc@microsoft.com.*

Thanks to the following technical experts for reviewing this article: *Paul Koch, Dan Liebling, Ann Loomis Thompson and Shane Williams*

DEVELOPED FOR INTUITIVE USE

DynamicPDF—Comprehensive PDF Solutions for .NET Developers

ceTe Software's DynamicPDF products provide real-time PDF generation, manipulation, conversion, printing, viewing, and much more. Providing the best of both worlds, the object models are extremely flexible but still supply the rich features you need as a developer. Reliable and efficient, the high-performance software is easy to learn and use. If you do encounter a question with any of our components, simply contact ceTe Software's readily available, industry-leading support team.

DynamicPDF

WWW.DYNAMICPDF.COM

TRY OUR PDF SOLUTIONS FREE TODAY!

www.DynamicPDF.com/eval or call 800.631.5006 | +1 410.772.8620

ceTesoftware

TED NEWARD

Building Combinators

In my December column (msdn.microsoft.com/magazine/hh580742), I looked at parser combinators, text parsers that are created by combining small, atomic parsing functions into larger functions, and those in turn into even larger functions, suitable for parsing non-trivial text files and streams. This is an interesting technique, one that builds on some core functional concepts, and it deserves deeper exploration.

Readers of the earlier column will recall that the parser we constructed to handle U.S.-style phone numbers worked, but the implementation was a bit … shall we say … quirky in places. In particular, the syntax for parsing three- and four-digit combinations—well, to be honest, it clunked. It worked, but it was hardly pretty, elegant or in any way scalable.

As a refresher, here's the Phone Number parser code:

```
public static Parser<PhoneNumber> phoneParser =
  (from areaCode in areaCodeParser
   from _1 in Parse.WhiteSpace.Many().Text()
   from prefix in threeNumberParser
   from _2 in (Parse.WhiteSpace.Many().Text()).
        Or(Parse.Char('-').Many())
   from line in fourNumberParser
   select new PhoneNumber() { AreaCode=areaCode, Prefix=prefix, Line=line });
```

The PhoneNumber type is pretty guessable. **Figure 1** shows the threeNumberParser and fourNumberParser, which, in particular, are what "clunk" with all the grace of a pro football defensive lineman attempting ballet for the first time on a stage greased with duck fat.

This is hardly the kind of inspirational coding practice that I hope to convey within these pages. There's a more elegant way to construct these, but to describe it, we need to dive a bit deeper into how parser combinators work. And that's the subject of this month's column. Not just because we need a more elegant way to construct parsers, mind you, but because the general technique helps describe how they work, and more important, how you might construct something like this in the future.

From Functions to Functions

The important point to realize about parser combinators is that a parser is really "just" a function: The function parses the text and may then transform the characters into something else. What that something else turns out to be is, of course, up to the person implementing the parser. It could be an abstract syntax tree (AST) for validation and verification of the text passed in (and later conversion into executable code or perhaps interpreted directly, as in some languages), or it could be a simple domain object, or even just values plugged into an existing class, like a dictionary of name-value pairs.

Figure 1 **Clunky Parsers**

```
public static Parser<string> threeNumParser =
  Parse.Numeric.Then(first =>
    Parse.Numeric.Then(second =>
      Parse.Numeric.Then(third =>
        Parse.Return("" + first.ToString() +
          second.ToString() + third.ToString()))));
public static Parser<string> fourNumParser =
  Parse.Numeric.Then(first =>
    Parse.Numeric.Then(second =>
      Parse.Numeric.Then(third =>
        Parse.Numeric.Then(fourth =>
          Parse.Return("" + first.ToString() +
            second.ToString() + third.ToString() +
            fourth.ToString())))));
```

Speaking in code, then, a parser looks like the following:

```
T Parse<T>(string input);
```

In other words, a parser is a generic function, taking strings and returning an instance of something.

As simple as that is, though, it's not entirely accurate. Were that the sum total of the story, we'd be back to writing a complete parser per function, which doesn't really allow for much in the way of reuse. But if we look at parsing as a series of functions—in other words, a parser is made up of a bunch of little parsers, each of which knows how to parse just a piece of the input and return just a piece of the resulting object—it's clear we need to return not only the resulting object, but also the remaining text that requires parsing. And that means the "T" from the previous declaration has to be made slightly more complicated by wrapping it in a "Parser Result" type that contains both "T" and a string with the remaining parse text, like so:

```
public class ParseResult<T>
{
  public readonly T Result;
  public readonly string Rest;
  public ParseResult(T r, string i) { this.Result = r; this.Rest = i; }
}
```

And given that C# naturally manages functions as delegate types and instances, declaring a parser now becomes a delegate declaration:

```
public delegate ParseResult<T> ParseFn<T>(string input);
```

Now, we can imagine writing a series of small parsers that know how to parse text into some useful other type, such as a ParseFn<int> that takes a string and returns an int (see **Figure 2**), or a ParseFn<string> that parses up to the first whitespace character, and so on.

Note that the parser implementation here is actually one that's pretty repeatable: to write a parser that parses text up to a whitespace character, all you'd need to do is change the IsDigit call to an

Figure 2 **Parsing a String and Returning an Int**

```
ParseFn<int> parseInt = delegate(string str)
  {
    // Peel off just numbers
    int numCount = 0;
    foreach (char ch in str)
    {
      if (Char.IsDigit(ch))
        numCount++;
      else
        break;
    }

    // If the string contains no numbers, bail
    if (numCount == 0)
        return null;
    else
    {
      string toBeParsed = str.Substring(0, numCount);
      return new ParseResult<int>(
        Int32.Parse(toBeParsed), str.Substring(numCount));
    }
  };
Assert.AreEqual(12, parseInt("12").Result);
```

IsLetter call. This screams for a refactoring to use the Predicate<T> type to create an even more fundamental parser, but that's an optimization we won't attempt here.

This implementation is great for parsing little things such as integers and single words, but so far it doesn't seem like too much of an improvement over the earlier version. This is more powerful, however, because you can combine functions by creating functions that take functions and return functions. These are called *higher-order functions*; while the theory is beyond the scope of this article, showing how they apply in this particular case isn't. The starting point is when you create functions that know how to take two parser functions and combine them in a Boolean "AND" and "OR" fashion:

```
public static class ParseFnExtensions
{
  public static ParseFn<T> OR<T>(this ParseFn<T> parser1, ParseFn<T> parser2)
  {
    return input => parser1(input) ?? parser2(input);
  }
  public static ParseFn<T2> AND<T1, T2>(this ParseFn<T1> p1, ParseFn<T2> p2)
  {
    return input => p2(p1(input).Rest);
  }
}
```

Both of these are provided as extension methods on the ParseFn delegate type to allow for an "infix" or "fluent interface" style of coding, to make it more readable in the end, on the theory that "parserA.OR(parserB)" reads better than "OR(parserA, parserB)."

From Functions to LINQ

Before we leave this set of small examples, let's take one more step and create three methods, as shown in **Figure 3**, that will essentially give the parser the ability to hook into LINQ, to provide a unique experience when writing code (this is one of Sprache's features as well). The LINQ libraries and syntax are in close sync with one another, in that the LINQ syntax ("from foo in bar select quux q …") is closely tied to the expectation that several method signatures are present and available for use. Specifically, if a class provides the Select, SelectMany and Where methods, then LINQ syntax can be used with them.

This gives LINQ the necessary methods to parse LINQ expressions, such as what you saw in the previous article.

Figure 3 **Where, Select and SelectMany Methods**

```
public static class ParseFnExtensions {
  public static ParseFn<T> Where<T>(
    this ParseFn<T> parser,
    Func<T, bool> pred)
  {
    return input => {
      var res = parser(input);
      if (res == null || !pred(res.Result)) return null;
      return res;
     };
  }
  public static ParseFn<T2> Select<T, T2>(
    this ParseFn<T> parser,
    Func<T, T2> selector)
  {
    return input => {
      var res = parser(input);
      if (res == null) return null;
      return new ParseResult<T2>(selector(res.Result),res.Rest);
     };
  }
  public static ParseFn<T2> SelectMany<T, TIntermediate, T2>(
    this ParseFn<T> parser,
    Func<T, ParseFn<TIntermediate>> selector,
    Func<T, TIntermediate, T2> projector)
  {
    return input => {
      var res = parser(input);
      if (res == null) return null;
      var val = res.Result;
      var res2 = selector(val)(res.Rest);
      if (res2 == null) return null;
      return new ParseResult<T2>(projector(val, res2.Result),res2.Rest);
     };
  }
}
```

I don't want to go through the exercise of (re)designing a parser combinator library here; both Luke Hoban and Brian McNamara have excellent blog posts on the subject (bit.ly/ctWfU0 and bit.ly/f2geNy, respectively), which, I must point out, serve as the standard against which this column is being written. I want only to demonstrate the mechanism by which these kinds of parsers are constructed in a parser combinator library like Sprache, because that provides the core of the solution to the earlier problem of the three- and four-digit parsers in the phone number parser. In short, we need one parser combinator that reads exactly three digits, and another that reads exactly four digits.

Figure 4 **Definition of the Many Combinator**

```
public static Parser<IEnumerable<T>> Many<T>(this Parser<T> parser)
{
  if (parser == null) throw new ArgumentNullException("parser");

  return i =>
  {
    var remainder = i;
    var result = new List<T>();
    var r = parser(i);
    while (r is ISuccess<T>)
    {
      var s = r as ISuccess<T>;
      if (remainder == s.Remainder)
          break;

      result.Add(s.Result);
      remainder = s.Remainder;
      r = parser(remainder);
    }

    return new Success<IEnumerable<T>>(result, remainder);
  };
}
```

Specifice

Given that the problem is to read precisely three digits and precisely four digits, it stands to reason that we want a function that reads exactly that number of characters from the input stream. The Sprache library doesn't give us that kind of combinator—there's a combinator that will read a repeating sequence of whatever-kind-of-character until it runs out of that-kind-of-character, but that's a "zero-to-many" (hence its name, Many) production rule, not a specific-number-of-characters rule, and thus unhelpful. Looking at its definition can be interesting and insightful, however, as **Figure 4** shows.

Figure 5 **The Twice Function**

```
public static Parser<IEnumerable<T>> Twice<T>(this Parser<T> parser)
{
  if (parser == null) throw new ArgumentNullException("parser");

  return i =>
  {
    var remainder = i;
    var result = new List<T>();
    var r = parser(i);
    var c = 0;
    while (c < 2 && r is ISuccess<T>)
    {
      var s = r as ISuccess<T>;
      if (remainder == s.Remainder)
          break;

      result.Add(s.Result);
      remainder = s.Remainder;
      r = parser(remainder);

      c++;
    }

    return new Success<IEnumerable<T>>(result, remainder);
  };
}
```

Figure 6 **The Specifice Method**

```
public static Parser<IEnumerable<T>> Twice<T>(this Parser<T> parser) {
  return Specifice(parser, 2); }
public static Parser<IEnumerable<T>> Thrice<T>(this Parser<T> parser) {
  return Specifice(parser, 3); }
public static Parser<IEnumerable<T>> Quadrice<T>(this Parser<T> parser) {
  return Specifice(parser, 4);
}
public static Parser<IEnumerable<T>> Quince<T>(this Parser<T> parser) {
  return Specifice(parser, 5);
}
public static Parser<IEnumerable<T>> Specifice<T>(this Parser<T> parser, int ct)
{
  if (parser == null) throw new ArgumentNullException("parser");

  return i =>
  {
    var remainder = i;
    var result = new List<T>();
    var r = parser(i);
    var c = 0;
    while (c < ct && r is ISuccess<T>)
    {
      var s = r as ISuccess<T>;
      if (remainder == s.Remainder)
          break;

      result.Add(s.Result);
      remainder = s.Remainder;
      r = parser(remainder);

      c++;
    }

    return new Success<IEnumerable<T>>(result, remainder);
  };
}
```

For most developers, the hardest part of this method (and, indeed, the entire Sprache library) is that the return value from this function is a function—specifically, a lambda method (always a Parser<> of some form, remember) that takes the string and returns an IEnumerable<T> in its result structure; the meat of the actual parser is buried inside that returned function, which means it will be executed later, not now. This is a long way from merely returning a T!

Once that oddity is dealt with, the rest of the function is pretty clear: imperatively, we step through and call the passed-in Parser<T> to parse each "whatever"; and so long as the parser keeps returning successfully, we keep looping and adding the parsed results to a List<T> that gets returned when everything completes. This serves as the template for my extension to the library, which I'll call Twice for now, essentially executing a given Parser<T> twice (and yielding an error if either parse fails), as shown in **Figure 5**.

In fact, while it would have been a bit easier to write this code by unrolling the loop-of-two into just two sets of imperative statements, remember, Twice isn't specifically what we're looking for. We need Thrice and Quadrice, and those are just special-case versions of Twice, with "3" and "4" instead of "2" in the code, which sounds as though we can extract them into a single method that takes the number of times to parse. I choose to call this method "Specifice," because we're parsing a specific number of times (see **Figure 6**).

Thus, we have now extended Sprache to parse exactly "ct" number of parses (characters, digits, whatever kind of Parser<T> we pass in), which opens up Sprache for use in fixed-length parsing scenarios, such as the ubiquitous fixed-length record text file, aka the flat file.

A Functional Approach

Sprache is a parser library for midsize parsing projects, for which regular expressions are too complex and a full-blown parser generator (such as ANTLR or lex/yacc) is overkill. It's not perfect, in that parser failures generate error messages that can be difficult to understand, but once you've gotten through the initial complexities, Sprache can be a useful tool in your developer toolbox.

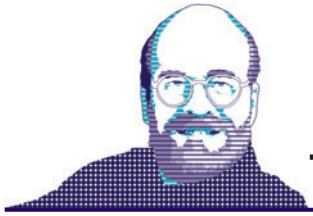
More than that, though, Sprache demonstrates some of the power and capability offered by a different approach to programming than what we're used to—in this case, from the functional world. So next time somebody asks you, "What good comes from learning about other languages if you aren't going to use them on the job?" there's an easy response: "Even if you aren't using a language directly, it can teach you techniques and ideas you can use."

But that's it for now. Next month, we'll take a stab at something entirely different. Because there's only so much concept and theory a language columnist's audience can take at once.

Happy coding! ■

TED NEWARD *is an architectural consultant with Neudesic LLC. He's written more than 100 articles, is a C# MVP and INETA speaker and has authored and coauthored a dozen books, including the recently released "Professional F# 2.0" (Wrox). He consults and mentors regularly. Reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

THANKS to the following technical expert for reviewing this article: *Nicholas Blumhardt*



CHARLES PETZOLD

Playing Audio Files in Windows Phone

When I first read that the enhancements in Windows Phone OS 7.1 included a way for applications to play sound and music files in the background, I thought, "Don't we have that already?"

It turns out I was correct, but only a little. It's indeed possible for a Windows Phone OS 7.0 application to play a music file in the background, but only in a very special case. In all the other cases, any music file that your Windows Phone OS 7.0 application plays will stop when your application is moved to the background. Of course, for most applications, this behavior is entirely appropriate and probably exactly what you want.

But consider an application that delivers music to your phone apart from the phone's normal music library. For such an application, it's extremely desirable to continue playing while other applications occupy the foreground or when the screen times out and goes into a locked state. And even for those of us who don't have a need to write such an application, this facility provides a fun entry point into exploring the new world of "background agents" introduced in Windows Phone OS 7.1.

In the next issue, I'll show you how to write a Windows Phone program that plays music files in the background. But to provide a broader picture of audio facilities on Windows Phone, I want to begin in this column with the more standard ways to play audio files supported in Windows Phone OS 7.0 as well as version 7.1.

MediaElement and Its Sources

The most common way for a Silverlight program to play a music or sound file is with MediaElement. Nothing is simpler: MediaElement derives from FrameworkElement, so you can put it in the visual tree of a XAML file and just set the Source property to a URL:

```
<MediaElement Source="http://www.SomeWebSite.com/CoolSong.mp3" />
```

When the XAML file is loaded, the music file automatically starts playing. MediaElement supports MP3, WMA and WAV files. Details are documented at msdn.microsoft.com/library/ff462087(VS.92).

As an alternative to referencing a file over the Internet, you can embed a sound or music file in your application executable. Add the file to the program in Visual Studio and flag the Build Action as either Content or Resource. (Content is preferred and embeds the file in the XAP executable; with Resource, the file is embedded in the DLL for the program.) Set the Source property to a URL referencing the file name with a folder name if applicable:

```
<MediaElement Source="Music/LocalSong.wma" />
```

Although MediaElement can be very simple, there are numerous ways to make it more complicated. One way is to specify the audio file at run time, as I did in the MediaElementDemo program, which is part of the downloadable code for this article.

In this program, the MediaElement is still in the visual tree, but the Source property isn't set, and AutoPlay is set to False. MediaElementDemo lets you play the three movements of the Brahms Violin Concerto. (The files are from the Internet Archive at archive.org/details/BrahmsViolinConcerto-Heifetz. It's a 1939 performance with violinist Jascha Heifetz and Serge Koussevitzky conducting, originally

Figure 1 **Downloading a Web File to Isolated Storage**

```
public MainPage()
{
  InitializeComponent();
  // ...

  // Check if file is in Isolated Storage; otherwise start downloading it
  using (IsolatedStorageFile isoStore =
    IsolatedStorageFile.GetUserStoreForApplication())
  {
    if (isoStore.FileExists(isoStoreRadioButton.Tag as string))
    {
      isoStoreRadioButton.IsEnabled = true;
    }
    else
    {
      WebClient webClient = new WebClient();
      webClient.OpenReadCompleted += OnWebClientOpenReadCompleted;
      webClient.OpenReadAsync(new Uri("http://www.archive.org/....mp3"));
    }
  }
  // ...
}

// When the music file is downloaded, save it to Isolated Storage
void OnWebClientOpenReadCompleted(object sender,
  OpenReadCompletedEventArgs args)
{
  if (!args.Cancelled && args.Error == null)
  {
    Stream inpStream = args.Result;
    byte[] buffer = new byte[inpStream.Length];
    inpStream.Read(buffer, 0, buffer.Length);

    using (IsolatedStorageFile isoStore =
      IsolatedStorageFile.GetUserStoreForApplication())
    {
      string isoPathName = isoStoreRadioButton.Tag as string;
      string isoDirName = Path.GetDirectoryName(isoPathName);

      if (!isoStore.DirectoryExists(isoDirName))
      {
        isoStore.CreateDirectory(isoDirName);
      }

      using (IsolatedStorageFileStream isoStream =
        isoStore.CreateFile(isoPathName))
      {
        isoStream.Write(buffer, 0, buffer.Length);
        isoStoreRadioButton.IsEnabled = true;
      }
    }
  }
}
```

Code download available at code.msdn.microsoft.com/mag201201TouchAndGo

Figure 2 **Setting the Source on MediaElement**

```
void OnRadioButtonChecked(object sender, RoutedEventArgs args)
{
  RadioButton radioButton = sender as RadioButton;
  string uriString = radioButton.Tag as string;

  // Save index for tombstoning
  radioButtonIndex = radioButtonPanel.Children.IndexOf(radioButton);

  if (radioButton == isoStoreRadioButton)
  {
    // Call SetSource on MediaElement using Isolated Storage stream.
    using (IsolatedStorageFile storage =
      IsolatedStorageFile.GetUserStoreForApplication())
    {
      using (Stream isoStream = storage.OpenFile(uriString, FileMode.Open))
      {
        mediaElement.SetSource(isoStream);
      }
    }
  }
  else
  {
    // Set Source property on MediaElement using URI
    Uri uri = new Uri(uriString, uriString.Contains(':')
      ? UriKind.Absolute : UriKind.Relative);
    mediaElement.Source = uri;
  }
}
```

available on Victor 78-rpm disks.) Three RadioButton elements have their Tag properties set to the sources of the three music files. For the first RadioButton, that's the full URL of the music file on the Internet Archive Web site. For the second movement, I downloaded the music file (named 02Ii.Adagio.mp3) to my PC, created a folder named Music in the project in Visual Studio and added that file to the folder. The second RadioButton references that file with the name "Music/02Ii.Adagio.mp3." When either of these two buttons is checked, the event handler obtains the Tag property and creates a Uri object out of it (specifying UriKind.Absolute for the Web reference and UriKind.Relative for the content) and sets that to the Source property of the MediaElement.

The second movement is a file of about 4.5MB, and obviously it increases the size of the executable by a considerable bulk. Adding files of this size to your executable is *not* recommended and done here only for demonstration!

If your application needs files of that size, a possible compromise is available: The application could download the file once over the Internet and save it to Isolated Storage. That's what I've done for the third movement of the Violin Concerto. The third RadioButton (which is assigned a name of "isoStoreRadioButton") has its IsEnabled property initially set to false. **Figure 1** shows the download process. In the page's constructor, if the file isn't in Isolated Storage, WebClient initiates a background transfer. When the transfer completes, the file is saved to Isolated Storage and the RadioButton is enabled.

Figure 3 **The MediaElementDemo Program**

In some contexts on Windows Phone OS 7.1, you can define a URI with a prefix of "isostore" to reference a file in Isolated Storage, but this doesn't work for MediaElement. Fortunately, MediaElement has a SetSource property that accepts a Stream object. **Figure 2** shows how the Checked handler for the RadioButton elements handles these differences.

Transports and Tombstones

Another way you can make MediaElement more difficult for yourself is by adding controls to pause and to move to the beginning or end of the file. Even more fun is a Slider control that lets you move to a particular point in the file, as shown in **Figure 3**.

The four ApplicationBar buttons are implemented very simply. Respectively, they set the Position property of MediaElement to zero, call the Play method of MediaElement, call the Pause method and set the Position property to the NaturalDuration property.

The tricky part is enabling and disabling the buttons. For that job, the CurrentStateChanged event of MediaElement is handled. When working out MediaElement logic, it's helpful first to use Debug.WriteLine in the event handler to get a feel for how the CurrentState property changes as a music file is loaded, buffered, played, paused and ended.

On the phone, all music and sound files are played through a single piece of software and hardware called the Zune Media Queue. If you use the standard Music+Videos application on the phone to play a song or album from your music collection, that music will continue to play in the background when you leave that application and start up other applications—and even when you start the MediaElementDemo program. However, if you start one of the movements of the Brahms Violin Concerto playing, the background music will stop. Now MediaElementDemo is in control.

But if MediaElementDemo leaves the foreground—whether by the user pressing the Start button or letting the screen time out—the Brahms will stop, even if the program is not tombstoned.

In such a circumstance, what do you want to happen when the user returns to the program? If the answer is "Nothing," you're in luck! But if you want the music to start up again from where it left off, MediaElementDemo demonstrates how this can be done. In its OnNavigatedFrom override, the program saves the index of the movement currently playing, the state (probably Playing or Paused) and the position. In OnNavigatedTo, the program checks the RadioButton and sets the state and position in the MediaOpened handler.

MediaLibrary and MediaPlayer

I mentioned that prior to Windows Phone OS 7.1, a facility already existed to play certain music files on the phone in the background. The catch is that these music files must be part of the phone's music library. Your program can

play one of these songs, or it can play all the songs on an album or all the songs of a particular artist or genre, or all the songs in a playlist.

The classes to do this are members of the Microsoft.Xna.Framework.Media namespace. To use these XNA classes in a Silverlight project for the phone, you first need to add a reference to the Microsoft.Xna.Framework library. With Windows Phone OS 7.0, Visual Studio gave you a warning about doing this. That warning is gone with Windows Phone OS 7.1.

Any Silverlight program that uses XNA classes to play music must include a special class that implements IApplicationService and calls FrameworkDispatcher.Update every 30th of a second. You can give that class any name you want, but you'll reference it in the App.xaml file in the ApplicationLifetimeObjects section:

```
<local:XnaFrameworkDispatcherService />
```

To play a song from the user's music library, start by instantiating the MusicLibrary class. Properties named Artists, Albums, Genres and Playlists provide collections of objects of type Artist, Album, Genre and Playlist, and all these classes include a Songs property of type SongCollection that's a collection of Song objects. (These collections are read-only; your application can't add anything to the user's music library or modify it in any way.)

To start playing something, use members of the static MediaPlayer class. The MediaPlayer.Play method accepts a Song object, a SongCollection or a SongCollection with an index to indicate the song to begin.

The PlayRandomSong program contains a button labeled "Play Random Song," and when you tap that, the following code executes:

```
void OnButtonClick(object sender, RoutedEventArgs args)
{
    MediaLibrary mediaLib = new MediaLibrary();
    AlbumCollection albums = mediaLib.Albums;
    Album album = albums[random.Next(albums.Count)];
    SongCollection songs = mediaLib.Songs;
    Song song = songs[random.Next(songs.Count)];
    MediaPlayer.Play(song);
}
```

This code extracts a random album from your collection, a random song from that album and starts playing it. (The Windows Phone emulator contains an album with a few tiny song files, so this program runs on the emulator just fine.)

If you start a song playing with PlayRandomSong, you'll find that you can navigate away from the program or even terminate the program and the song will continue playing. It's exactly as if you played that song from the phone's regular Music+Videos application—and if you start that application, you'll see the album cover and the song's title. Moreover, if you press the volume control button on the phone, you'll see the song at the top of the screen and get access to buttons to pause or go to the beginning or end of the song.

Just as the phone's Music+Videos application knows what song you've played with MediaPlayer.Play, your application can determine which song the phone's Music+Videos application is currently playing. This information is available from the Queue property of MediaPlayer, which provides a MediaQueue object that indicates the song currently playing and a collection of songs if an album or playlist is playing. The PlayRandomSong program uses a timer to check the ActiveSong property of the queue and displays information about that song. Alternatively, you can set handlers for the ActiveSongChanged event of MediaPlayer.

Creating Song Objects

The PlayRandomSong program obtains a Song object from one of the properties or collections of MediaLibrary, but Song also has a static property named FromUri that creates a Song property based on a file not in your music library. This URI can reference a music file over the Internet or that's part of the program's XAP file. (It can't reference a file in Isolated Storage.) You can then use MediaPlayer to play this Song object. (You can't create your own SongCollection objects.)

The MediaPlayerDemo program shows how it's done. This program let you play the Brahms Double Concerto (another 1939 recording from archive.org/details/BrahmsDoubleConcerto_339) with Heifetz again, Emanuel Feuermann on cello and Eugene Ormandy conducting. Because you can't use MediaPlayer with Isolated Storage, both the first and last movements are Web references.

Another difference is that the Position property of MediaElement is both gettable and settable, while the PlayPosition property of MediaPlayer is only gettable. Consequently, the two ApplicationBar buttons that go to the beginning and end of the track aren't applicable. Also, there's apparently no way to get the duration of a Song object created in this way, so the Slider is irrelevant as well. I've also removed all the tombstoning logic from this program because it's not possible to start up a track where you left off.

Because MediaPlayer plays a Song object obtained from the phone's music library in the background, you might expect it also to play *any* Song object in the background. It does not. In this respect, MediaPlayer is just like MediaElement. The music stops as soon as you navigate away from the application. However, if you navigate away from the MediaPlayerDemo program and it's not tombstoned—which often happens with Windows Phone OS 7.1—the music is only suspended. When you navigate back to the application, it picks up where it left off.

I think when you chalk up the pluses and minuses, the Silverlight MediaElement is a little ahead of the XNA MediaPlayer, but the automatic resumption of playback is a very nice MediaPlayer feature.

Streaming and Beyond

I've been discussing playing common audio and music files in WMA and MP3 formats. Windows Phone also allows a program to generate sound dynamically within the application. In the context of Windows Phone programming, this is known as "streaming." I demonstrated one approach in the SpeakMemo program in my February 2011 UI Frontiers column (msdn.microsoft.com/magazine/gg598930) using the XNA DyanamicSoundEffectInstance class. You can also use the MediaStreamSource class in Silverlight to do something similar. This is how to implement electronic music synthesis on the phone. Again, however, these are only usable by foreground applications.

Beginning in Windows Phone OS 7.1, the concept of a "background agent" has been introduced, and you can use this for playing either music files or streaming audio while your program is suspended in the background.

In the next issue, I'll discuss how this is done. ■

CHARLES PETZOLD *is a longtime contributing editor to* MSDN Magazine. *His Web site is charlespetzold.com.*

THANKS to the following technical expert for reviewing this article: *Mark Hopkins*

DAVID PLATT

Lowering Higher Education

I have a front-row seat to the coming revolution. One of the main services I provide to Microsoft and the developer community is bridging the divide between them and academia. As with newspapers, I expect the existing order in academia will be redefined over the next decade, posing enormous challenges and huge new market opportunities for the institutions and people who can grab them.

At the source of the coming revolution is this simple fact: The inflation-adjusted price of college has quadrupled since 1982 (source: cnnmon.ie/bApHLC). Has the value of that education quadrupled, or even doubled? Not that I can see. That artificial price increase has created an academic bubble like the stock and real estate bubbles we've encountered recently. Now combine disruptive technology (ubiquitous fast Internet) with the worst economy in living memory and you spark off cataclysmic structural change. The bubble is about to burst.

> Think back to your freshman calculus class. Was your instructor any good? Every person that I interviewed for this column swore that he'd gotten the world's worst.

Some academic institutions are already adapting to this new reality, such as the Harvard University Extension School (extension.harvard.edu), where I have the honor of teaching. We admit everyone, and an undergraduate course costs about $1,000. Many of our courses, including mine on .NET, are available over the Web. You won't play as much Frisbee or drink as much beer (probably) as in a classic residential college, but how many parents and students today wouldn't find those economics compelling? And I'll stake our teaching quality against any challenger, anywhere, any time.

But this model still relies on human instructors delivering live lectures, increasing cost. That's about to change.

Think back to your freshman calculus class. Was your instructor any good? Every person that I interviewed for this column swore that he'd gotten the world's worst. I know I did. (Yes, Sue Esch and Robert Nelson, I mean you.)

Imagine taking the world's 10 best teachers of freshman calculus and paying them each a million dollars for a video. Put them all online at $100 a pop, including exams and problem sets. Capture just 3 percent of the roughly 4 million college freshmen in the United States, and you've recouped your investment in the first year.

Wouldn't students prefer that to paying $12,500 (one-fourth of their yearly bill at a private university) to suffer through an apathetic graduate student boring everyone in a 500-seat lecture hall? Wouldn't they rather enjoy a far better instructor, watch on their own schedules, re-run confusing sections until they understood them, progress at their own paces, for less than 1 percent of the price? Not just yes, but *hell, yes*!

Now expand this idea to any large lecture class where the material seldom changes: freshman economics, organic chemistry, English literature, even introductory computer science. It won't cover everything, but looking back at my own college transcript (painful), it could have replaced about three-quarters of my classes. Quality way, way up; price way, way down.

This leveraged approach can work even for hot current topics. Stanford University announced that its class on artificial intelligence will be available for free online, and 58,000 students registered to take it. I might sign up myself.

This model has some hurdles, but with such huge cost advantages, they'll get solved. For example, the Stanford online attendees won't get grades or credit. How long until an enterprising community college offers an exam on the content and credit for successful completion, priced at perhaps $500? Students connecting to other students? Some entrepreneur will open an academic bar, like a sports bar, serving beer with your calculus videos. Asking the instructor questions? We'll think of something.

The biggest loss would be those few extraordinary teachers who show you the world in a new light. I was lucky to study under Vic Mansfield, who influences my teaching and even my basic thinking to this day. I pay him the supreme compliment, not of imitation, which he would consider a lower form, but of adaptation, taking what he taught me and making it mine before passing it on. (And so do my classmates. See Vic's obituary at bit.ly/k3qK4A, particularly the comments.) Perhaps we'll develop some form of mentor classes to fill this gap.

As with all technological advances since fire and the wheel, those who cling blindly to the old ways get trampled. The faculty and institutions that prosper will be those that recognize the coming changes and adapt to them early (see my May and June columns from last year), instead of struggling, futilely, to hold back the tide. I expect to find the landscape radically different when my daughters start college, nine years from now. ■

DAVID S. PLATT *teaches Programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

Richard **Campbell**

Carl **Franklin**

We Are Smart Developers

New Version!
SPREAD.NET 6 Professional

"What I want is a way to take all that Excel goodness and plop it into my .NET application. Welcome to GrapeCity Spread."

Carl **Franklin**
Host, .NET ROCKS!

Smarter Components for
SMARTER DEVELOPERS

GrapeCity PowerTools

www.GCPowerTools.com
GvTv.GCPowerTools.com

SPREAD.NET
GrapeCity PowerTools

Now includes MultiRow!

Embedded Spreadsheet Platform
for Advanced Business, Engineering
and Scientific Applications

Start Building Smarter Applications!

2010-11 Componentsource Top 100 Product

SD Times 2011 100

Photograph: © 2011 F&S Photography

If you are short on either part of this equation you should TALK to Us

Syncfusion®
• NET Componets

www.syncfusion.com/evaluation
30 Day Trial
Full Support